Power Fingerprinting for Integrity Assessment of Embedded Systems

Carlos R. Aguayo González

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical Engineering

Jeffrey H. Reed, Chair
William H. Tranter
Charles W. Bostian
Jung-Min "Jerry" Park
Stephen H. Edwards

December 5, 2011
Blacksburg, Virginia

Keywords: Power Fingerprinting, instrussion detection, pattern recognition, embedded systems, integrity assessment

Power Fingerprinting for Integrity Assessment of Embedded Systems

Carlos R. Aguayo González

(ABSTRACT)

This dissertation introduces Power Fingerprinting (PFP), a novel technique for assessing the execution integrity of embedded devices. A PFP monitor is an external device that captures the dynamic power consumption of a processor using fine-grained measurements at the clock-cycle level and applies anomaly detection techniques to determine whether the integrity of the system has been compromised. PFP uses a set of trusted signatures from the target code that are extracted during a pre-characterization process. PFP provides significant visibility into the internal execution status, making it extremely robust against evasion. Because of its independence and physical separation, PFP prevents attacks on the monitor itself and introduces minimal overhead on platforms with resource constraints. Due to its anomaly detection operation, PFP is effective against unknown (zero-day) attacks.

This dissertation demonstrates the feasibility of PFP on different platforms with different configurations and architectural complexities. Experimental results demonstrate the feasibility of PFP in a basic deterministic embedded platform for radio applications in two different areas: security and regulatory certification. For more complex, non-deterministic platforms, this works presents feasibility results for monitoring the execution integrity of complex software on a high-performance Android platform, including the ability to detect a real privilege escalation attack. In addition, the dissertation develops several general techniques to implement and integrate PFP into embedded platforms such as a general monitoring architecture, a methodology to characterize software modules and extract signatures, and an approach to perform board characterization and improve monitoring sensitivity.

To Lauren

Acknowledgments

I would like to give special thanks to my advisor Dr. Jeffrey H. Reed, who spent countless hours with me developing this technology. Your vision, guidance, and support made this project possible. I would also like to thank the rest of my committee, who have all contributed significantly to my research and education: Dr. Bill Tranter, Dr. Charles Bostian, Dr. Jung-Min Park, and Dr. Stephen Edwards. Thank you for your ideas, your valuable feedback, and your patience.

Very important throughout this journey have been all my fellow students at Wireless @ Virginia Tech. It has been a long and difficult ride, but our constant interactions made it very enjoyable. In particular, I want to thank "José" Gaeddert, Chris Phelps, Tom Tsou, Philip Balister, Max Robert, Chris Anderson, Chris Headley, Shereef Sayed, Ben Hilburn, and so many more that have enriched my life with our friendship. Also, I need to thank our staff: Jenny, Hilda, Nancy, Cyndy, and Shelby. No research would happen in this lab without your help.

I also have to thank my extremely supportive family. You have been there for me over all these years, providing me with support, direction, and encouragement. You are my role models and a constant source of inspiration, wisdom, and love.

And finally, I need to thank you, Lauren, for all your support, patience, motivation, love, and everything. This work would have never been possible without you.

Grant Information

This work was supported in part by the National Science Foundation under Grants No. 0627436 and 0910531, Wireless @ Virginia Tech Affiliates, Tektronix, and Texas Instruments. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, Wireless @ Virginia Tech Affiliates, Tektronix, or Texas Instruments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cyber systems capable of network connectivity have delivered unprecedented flexibility, agility, and effectiveness into modern technology, industry, and services. For example, communication systems are better able to adapt to changing environmental conditions and optimize the use of radio resources due to the introduction of software-defined radio (SDR) and cognitive radio (CR); industrial processes are more efficient because of automated control; and road vehicles are more efficient and safer due to a variety of intelligent software systems. Computer devices and software systems are used pervasively across industries, including critical applications such as power distribution, medical equipment, weapon systems, and many more.

As the importance, complexity, and reach of software systems keeps expanding, so does the difficulty in securing these systems from cyber attacks. Software systems, including relatively simple embedded devices, are inherently vulnerable to cyber attacks. With the increased diversity in embedded platforms and application the number of potential attack points grows and makes it relatively easy for an adversary to find new vulnerabilities and develop new attacks to disrupt critical embedded systems. In this document, we present a novel integrity assessment technique based on monitoring the power consumption of the processor to improve the security of embedded systems.

1.1 Integrity Assessment in Embedded Devices

Securing a cyber system is a complex system-level task that requires several technologies working together to provide system integrity, service availability, and information privacy. From these three areas, system integrity resides at the core of the cyber security strategy because without it a reasonable expectation of privacy, or availability, is not possible.

Several approaches have been developed to provide system integrity, including robust access control [63] and process isolation [5] in operating systems, tamper protection at the hardware level [7], and formal methods to eliminate exploitable software defects [76]. Against a sophisticated adversary, however, current approaches to counter cyber attacks are significantly outmatched, as demonstrated by the consistent reporting of new vulnerabilities and successful attacks. More importantly, the range of potential cyber targets keeps expanding, putting systems such as embedded systems, once thought to be safe, well within the rage of cyber attackers. For example, the recent Stuxnet [48] attack to industrial control systems targeted programmable logic controllers (a platform previously considered to be out of reach) capable of disrupting industrial operations and even causing physical damage [42]. Because traditional perimeter defenses in cyber security will never be perfect and attackers will eventually find a way to circumvent them—sometimes with the help of malicious insiders—critical systems need a reliable method to assess their operational integrity, detect the presence of security breaches, and trigger an appropriate response.

The most common integrity monitoring and intrusion detection approach is represented by an antivirus system. These systems scan the memory and peripherals looking for signatures from malicious software (malware) or activities. Unfortunately, current antivirus solutions depend on having explicit knowledge of the malware itself, which presents a challenge as new malware keeps being introduced and new attack vectors are developed. Furthermore, it is possible to evade detection from an antivirus system by modifying the signatures of the malware.

In the context of embedded devices, the problem of integrity monitoring is exacerbated by the large variety of platforms and operating systems. This is in part because existing antivirus systems are usually tied to a specific operating system and do not translate well to the application-specific platforms common in embedded systems. Furthermore, embedded platforms have severely constrained resources as well as stringent performance and reliability requirements. This leaves little room, in terms of processing and memory, to perform integrity assessment tasks.

In order to counter current and future cyber threats to critical embedded systems, it is necessary, first, to recognize that vulnerabilities will be present in spite of our best efforts to eliminate them and, second, to devise techniques that provide reliable system integrity assessment. With timely reliable information about the integrity status of a system, it is possible to respond to threats before they can cause damage and prevent the catastrophic scenario of trusting an already compromised device. In order to be effective in an embedded environment, such a technique must introduce minimal overhead, be capable of detecting zero-day attacks, and be universally applicable without being tied to a particular platform.

Different approaches to monitor the execution status of a processor have been developed. These approaches can be roughly classified in two categories: internal and external. Internal approaches, such as host-based intrusion detection systems (IDS), are implemented by software or special hardware on the host device. Internal approaches have excellent visibility of the execution status, as they reside within the processor space, but they also introduce overhead. External approaches, such as network-based IDSs, are implemented by separate devices that observe data interfaces and look for special communication patterns that indicate an intrusion. External approaches have reduced visibility of the execution status, but by virtue of being outside the processor scope, they do not add processing overhead. Along with the development of these monitoring approaches, sophisticated attack and evasion mechanisms have emerged. Host-based systems, are difficult to evade, because they reside within the processor and have high-visibility, but this also makes them vulnerable to attack. External approaches on the other hand, are difficult to attack but their visibility of

the execution status is reduced and makes them easy to evade.

There are, however, alternative mechanisms to obtain information about the internal execution status in a processor. These mechanisms involve the observation of different elements outside the processor scope that are affected by the current execution taking place. These elements, known as side channels, include temperature, power consumption, event timing, electromagnetic radiation, etc. These side channels have previously been exploited for malicious purposes, attacking the integrity or confidentiality of systems [27]. The same principles used to extract execution information and perform side-channel attacks, however, can be used to monitor the integrity of critical systems.

1.2 Thesis

A significant portion of the power consumption in a digital processor is dynamic and depends directly on the specific software being executed. Thus, dynamic power consumption provides a side channel that carries information about the execution status of a processor and can be used to determine whether it corresponds to the execution of specific software.

This document introduces a novel approach for integrity assessment of digital devices called Power Fingerprinting (PFP). PFP uses fine-grained side-channel information from the processor's dynamic power consumption to perform anomaly detection and determine whether the integrity of the system has been compromised. The reference signatures used by PFP capture information at the clock cycle level and allow an external monitor to observe the detailed execution status of a processor, even those with constrained resources. PFP relies on a pre-characterization process to extract unique signatures from the power consumption during the execution of the target module. During the monitoring process, signal detection and pattern recognition techniques are used to determine whether the observed power consumption and the signatures match.

The principles behind PFP apply to any digital system, but the scope of this research is

limited to embedded systems. Appying PFP to embedded devices enables security monitoring and integrity assessment on platforms that otherwise would not have the memory or processing resources necessary for it. The examples and experiments in this work are mostly related to SDR and CR applications, but the techniques and results are not limited to these applications. The feasibility demonstrations in this dissertation are performed by characterizing the power consumption during the execution of target routines at the bit-transition level, getting status information directly from the hardware, and successfully discriminating, in blind tests, between the execution of the original target routines and tampered versions.

1.3 Contributions

This document makes the following original contributions to the subject of integrity assessment in embedded systems:

1. The introduction of a novel approach to perform integrity assessment in embedded platforms called Power Fingerprinting (PFP)
2. A description of the methods and techniques that conform PFP technology
3. A general architecture for PFP
4. A mechanism to track and compensate for the impact environmental changes have on power observations (traces)
5. The methodology, including the necessary tools and techniques, to characterize software execution, extract signatures, and design optimal detectors
6. A general signaling approache for synchronization and module identification using physical signals
7. A general approach to embed synchronization and module identification signaling into the power traces

8. A demonstration of the feasibility of PFP on deterministic platforms

9. An implementation and validation of PFP monitor using commercial bench-top test equipment

10. A mechanism to characterize the power consumption of a given platform and its application to reduce the dimensions in the feature space and improve the sensitivity of the PFP monitor.

11. A general methodology to evaluate the minimum sensitivity of a PFP monitor to changes in the execution status by mathematically quantifying performance through a receiver operating characteristics model

12. A demonstration of the feasibility of PFP on non-deterministic platforms

13. An approach, description, and demonstration of a mechanism to characterize execution at the kernel level

14. A demonstration of the ability of a PFP monitor to detect a privilege escalation attack (Malware) as it manages to violate the application sandbox imposed by the Android platform

1.4 Organization

The organization of this work is as follows: Chapter 2 provides a brief introduction and background information on different technologies and concepts upon which PFP builds. The chapter explains power consumption in CMOS devices and introduces basic concepts of pattern recognition necessary to PFP. It also provides a brief description of the regulatory environment in SDR, along with a description of side-channel attacks, which share the same core principles with PFP. The chapter ends with a brief description of alternative mechanisms for integrity assessment.

Chapter 3 describes in detail the theory of operation of PFP. It describes the processes necessary to extract discriminatory features, characterize software execution and obtain unique signatures, design optimal detectors, and monitor the integrity of target systems. Chapter 4 describes the application of PFP to deterministic platforms, which are processors with basic architectures. The chapter presents the results of two experiments that demonstrate the ability of PPF to determine the integrity of basic platforms. The results show that a PFP monitor is capable of successfully detecting execution deviations that impact the security settings and spectral emissions in a basic radio platform. The chapter also describes a technique to improve the sensitivity of PFP by characterizing the specific way a processor consumes power.

Chapter 5 describes the application of PFP to non-deterministic platforms, which are processors with complex, sophisticated architectures. The chapter presents the results of two experiments that demonstrate the ability of PPF to determine the integrity of complex platforms. The results show that a PFP monitor is capable of successfully detecting conditional malicious execution in an Android platform. They also demonstrate the ability of PPF to detect the successful execution of a privilege escalation attack to the Android from malware found in the wild. Chapter 6 concludes with a brief discussion of advanced topics and extensions to the PFP theory and implementation provided here.

Chapter 2

Background

PFP observes the power consumption of a processor and then applies pattern recognition techniques to determine whether the observed power traces match the signatures. This chapter provides a high-level review for some of the technologies and concepts that support PFP. The chapter starts with a brief introduction to CMOS technology, the most popular technology for building digital processors. It continues with a review of basic pattern recognition and machine learning concepts.

Because PFP integrity assessment is also envisioned to have a significant impact in applications with strict regulatory requirements, such as radio communications and avionics, this chapter provides a brief overview of the current regulatory framework for SDR and CR. The chapter concludes with a review of related technologies and previous work that support PFP research, including: power analysis side-channel attacks and execution monitoring.

2.1 CMOS Technology

Complementary metal-oxide-semiconductor (CMOS) is a technology for designing and implementing integrated circuits. Due to its low power dissipation and noise immunity, CMOS

has become the dominant technology in digital processors manufacturing. The low power consumption of CMOS technology is because significant power drain occurs only during switching.

CMOS circuits use a combination of p-type and n-type metal-oxide-semiconductor field-effect transistors (MOSFETs) to implement logic gates. These transistors have a remarkable ability to behave almost like ideal switches, making CMOS circuit design practical and interesting. In order to implement the logic gates, the p- and n-channel MOSFETs (called PMOS and NMOS, respectively) are configured to create paths from the power source or ground to the output and create this way the desired logic functions, both simple and complex.

The NMOS transistor behaves as an open switch when the gate voltage S=0, and as a closed one when S=1. The PMOS transistor works in the opposite way: when S=0, the PMOS transistor behaves as a closed switch, and when S=1 as an open switch. Figure 2.1 depicts a simplistic approximation of the PMOS and CMOS transistor operation.

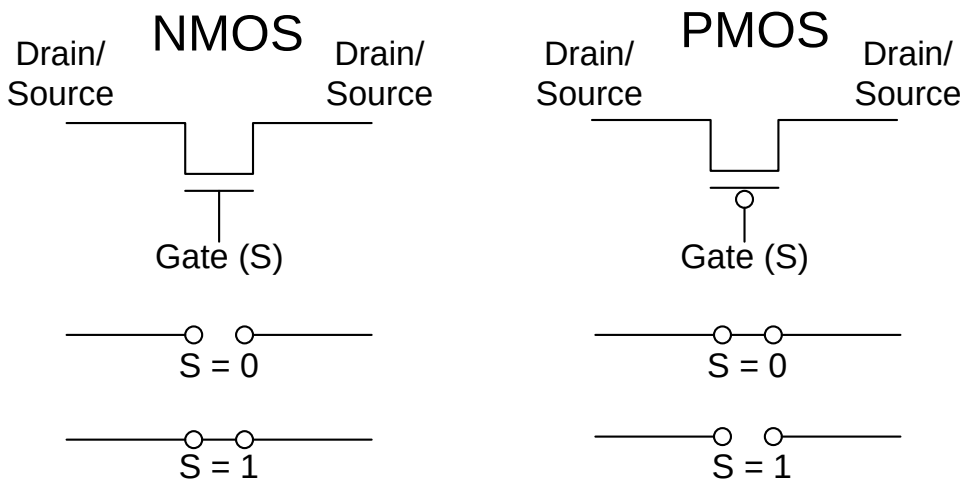

Figure 2.1: Simplified description of NMOS and PMOS transistors switching operation

PMOS and NMOS transistor pass high and low voltages in a different and peculiar way. It has been observed that the n-channel is very good at passing logical "0's" and the p-channel passes logical "1's" very well, which results in different configurations providing strong and weak logical states [52].

The power dissipation in CMOS circuits can be classified in two categories: static and dynamic. Static power dissipation which happens when the circuit is idle is caused by sub-threshold conditions when the transistors are off, tunneling, and reverse leakage currents. The dominant power dissipation, however, happens during the transitions in the circuit states. This dynamic power dissipation is the basis of PFP and is explained in more detail in the next chapter.

2.2 Pattern Recognition and Machine Learning

The recognition of a pattern, even as done by humans, is a process of *"estimating the relative odds that the input data can be associated with one of a set of known statistical populations which depend on past experience and which for the clues and the a priori information for recognition"* [70]. In this section we present a very brief introduction to the problem of pattern recognition. A more detailed treatment of the different theories and techniques can be found in [70, 25, 35].

The problem of pattern recognition can be defined as the categorization of input data into identifiable classes by means of extracting discriminatory features or attributes of the input data from all the irrelevant details. In general, the design of an automatic pattern recognition system requires three major tasks:

1. **Sensing**, which is the representation of measurable input data from the objects to be recognized in a format readable by the system. These measurements can be arranged in the form of a pattern vector:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

   For example, for a continuous signal $f(t)$, sampled at times $t_1, t_2, \ldots, t_n$, a pattern

vector can be formed by letting $x_1 = f(t_1), x_2 = f(t_2), \ldots, x_n = f(t_n)$. Pattern vectors contain all measured information about the patterns and it is useful to think of them as a point in an n-dimensional Euclidean space.

2. **Feature Extraction**, which involves extracting characteristic features from the input data and reducing the dimensionality of the pattern vectors. Features of a pattern class are the characteristic attributes common among all patterns in the class. These features are also known and intraset features. Differences between classes are known as discriminatory or interset features. Very often there are features that occur across all classes. These features carry no discriminatory information and should be discarded.

   If a complete set of discriminatory features exists, then the recognition problem is straight forward. Unfortunately, in practical applications this is extremely difficult and almost never the case. Hence, pattern recognition systems have to operate with incomplete information. There is no universal method for identifying the features that carry the most discriminatory information or to decide when the information is sufficient to achieve a specific performance metric. The selection process is greatly dependent on the application and environment from where the patterns are extracted.

3. **Classification**, which covers the determination of optimum decision procedures for assigning patterns into classes.

   Assuming that a system is designed to recognize M different pattern classes, denoted by $\omega_1, \omega_2, \ldots, \omega_M$, then the pattern space consists of M regions, each of which encloses the pattern points of a class. It follows that the classification problem can now be viewed as that of generating optimal decision boundaries to separate the M pattern classes. For example, let the decision boundaries be defined by decision functions $d_1(\mathbf{x}, d_2(\mathbf{x}, \ldots, d_M(\mathbf{x}$. These are scalar and single-valued functions of the pattern vector $\mathbf{x}$. Then the pattern $\mathbf{x}$ belongs to class ω_i if $d_i(\mathbf{x}) > d_j(\mathbf{x})$ for $i, j = 1, 2, \ldots, M$, and $i \neq j$.

2.3 SDR Regulatory Framework

Under the current regulatory framework, traditional radios are not allowed to modify their operating frequency, output power, or types of radio frequency emissions without going through a re-certification process. This is because such changes involve hardware modifications that practically yield a new device. With SDR, however, these changes can be made with just a software update. Therefore, regulatory bodies and manufacturers were forced to devise new policies, certification techniques, and enforcement mechanisms to maintain the delicate balance between interference management and regulatory burden.

Currently, the US Federal Communications Commission (FCC) has adopted new policies to regulate SDR. In the new rules, the FCC:

> *"amended its equipment authorization rules to permit equipment manufacturers to make changes in the frequency, power, and modulation parameters of such radios (SDR) without the need to file a new equipment authorization with the Commission. We (the FCC) will also permit electronic labeling so that a third party may modify a radio's technical parameters without having to return it to the manufacturer for relabeling"* [28].

This policy designates a new class of permissive changes. "*Any changes in frequency, power, or modulation type of a software defined radio may be authorized as a Class III permissive change*". This designation streamlines the filing procedure for changes to approved SDR and eliminates the need for new identification numbers. The manufacturer, however, still needs to submit test data showing that the equipment complies with the applicable rule parts and RF exposure requirements with the new software.

This policy, however, limits certification to specific hardware-software pairs to prevent unknown effects on the RF emissions of a radio. The only hardware changes allowed with a Class III permissive change are those in which the hardware modifications do not affect radio frequency emissions. **This policy also requires manufacturers to prevent unau-**

thorized software changes that could affect the compliance of a radio. This was never a concern with legacy radios. The FCC, however, stopped short of mandating specific requirements for this authentication. Instead, it is left to the manufacturers to "*take steps to ensure that only software that is part of hardware/software combination approved by the Commission can be loaded into a radio*". The Commission leaves open the possibility of specifying more detailed security requirements at a later date as SDR technology improves

This policy seems to be adequate for current systems. Unfortunately, there is no way to monitor the continuous compliance of devices over time. As SDR technology becomes more prevalent and becomes target of more malicious attacks, it will be necessary to continuously assess software execution integrity. Due to its ability to detect deviations from authorized execution, PFP can be a powerful tool in regulatory compliance monitoring.

2.4 Related PFP Work

Dynamic power consumption in digital processors has long been studied in different research areas, from low-power implementations of embedded systems [45, 23] to side-channel attacks to cryptographic devices [57, 41, 44, 61, 68]. Power monitoring in power efficiency research usually focuses on characterization and benchmarking of different instructions or types of instructions to model power consumption behavior of specific platforms. In terms of power analysis side-channel attacks, most of the published work focuses on observing dynamic power consumption of simple cryptographic devices in order to estimate the secret key. Side-channel attacks are explained in more detail in the following section.

Work by Buennemeyer, et.al. [20, 19] explores battery monitoring techniques to detect sleep deprivation and battery exhaustion attacks to hand-held devices. These techniques rely on monitoring instantaneous current drain, power management state information, and known attack signatures to provide an intrusion detection mechanism. They stop short, however, from providing execution status monitoring.

Power analysis has also been applied to detect hardware Trojans. For this application, the power consumption of VLSI chips is monitored during testing and compared against the power consumption of a trusted reference [72, 59]. During testing, random inputs are provided to trigger a partial activation of the potential Trojan. A tampered device will have a different power consumption level due to the additional bit transitions caused by the extra logic implementing the Trojan.

2.4.1 Power Analysis Side-Channel Attacks

PFP relies on the execution status information carried by the processor's dynamic power consumption. The same principle has been exploited in power analysis side-channel attacks to obtain secret cryptographic keys from smart cards [58, 41, 44].

Side-channel attacks were introduced in 1996 [40]. Their non-invasive nature allows them to attack a device during its normal operation without causing any physical harm. Side-channel attacks have been successfully used to extract cryptographic key material of symmetric and public key encryption algorithms, such as Data Encryption Standard (DES) and Advanced Encryption Standard (AES), running on microprocessors, DSPs, FPGAs, ASICs and high performance CPUs. Side channel attacks are commonly performed using power analysis [41, 33, 66, 54, 71], but they are also implemented observing changes in time delay [40, 24], electromagnetic radiation [30], or even fault behavior [16, 17].

Side-channel attacks based on power analysis have been attempted using a number of techniques that vary in terms of resources required, number of observations, and success rate. For example, Simple Power Analysis (SPA) involves directly interpreting power consumption measurements collected during cryptographic operations and can yield algorithm as well as key information [41]. Differential power analysis on the other hand, requires more involved statistical analysis of the power measurements, often by correlating the data being manipulated with the side channel information, to exploit specific biases in power consumption during cryptographic operation and obtain key values. In general, all power analysis

side-channel attacks rely on having knowledge of the encryption algorithm being employed, the input to the cryptographic device, a model of the device's power consumption, and power measurements from several encryption operations. Attackers use all this information to identify the cryptographic key value that is more likely to generate the observed power consumption give the specific model used.

Different from power analysis side-channel attacks, however, PFP does not attempt to reverse-engineer the executed code, but only to characterize its power consumption. Because SDR developers have control over the source code itself, the characterization task can be facilitated by developing software such that it yields tractable fingerprints by providing a conducive structure and potentially inserting strategic instructions to enhance certain power features.

2.4.2 Monitoring and Intrusion Detection

Real-time execution status monitoring is traditionally accomplished by inserting small software constructs and reporting tasks in the target system as in [62, 49]. These schemes, however, suffer from the observer's effect and depend on the same processor to perform the monitoring tasks, leaving the monitor itself vulnerable to attacks. In terms of integrity assessment, approaches such as the Trusted Computing Module (TCM) and Intel's Trusted Execution Technology have received significant attention. The former performs load time attestation using hash functions and secure key management and storage. The latter builds from the TCM and provides hardware-based protected execution and memory spaces. Unfortunately, the collision resistance of some hash functions has already been compromised [73]. Furthermore, these techniques only authenticate the loading of software, not its execution, leaving devices vulnerable to fault induction by unexpected environmental conditions.

Intrusion detection systems need to observe the execution status of the processor in order to detect malicious software installed. There are several monitoring techniques to perform intrusion detection studied in the literature, each one with its own advantages, limitations,

and intrinsic costs. For example, Virtual Machine Monitoring (VMM) [32] has been a popular technique that received a lot of interest. In VMM, a thin software layer running directly on the hardware abstracts the underlying platform and allows a hypervisor to monitor the low-level operating system calls and operations. VMM, unfortunately, has a performance and latency impact and is inherently visible from the attacker's perspective [31]. This information can be used by malicious attackers to tailor attack models. Other intrusion monitoring methods include: dynamic taint analysis [50], system call monitoring [65], and dynamic information flow tracking [67]. Each approach has its own strengths and weaknesses. For example, dynamic taint analysis provides fine-grained execution monitoring and has been successfully applied to a variety of software, but has been shown to be relatively easy to evade in some contexts [21]. In general, all these approaches provide different levels of visibility into the execution status which is traded off against complexity and isolation against attacks to the intrusion monitor itself. The best visibility is achieved by placing a monitor inside the processor, which makes the monitor vulnerable to attacks. Some network-based intrusion detection techniques take the monitor out of the processor, providing the required isolation to protect the monitor against attacks, but with diminished visibility, making the monitor easier to evade. PFP has the ability to monitor the actual execution of software by using a side channel. This approach allows an external monitor to achieve great visibility, making it difficult to evade, while maintaining the appropriate isolation to prevent attacks to the monitor itself.

Traditional intrusion detection systems rely on a database of malicious signatures to identify unauthorized intrusions. The signatures can be obtained directly from the malware payload or from general high-level behavior. The database needs to be updated every time new malware is created in order to maintain effective protection. This means that the database keeps growing and is always one step behind potential attackers. When signatures are extracted from the malware payload, the resulting monitor can be evaded by simple obfuscation, polymorphism, and packing techniques, and thus requiring the database to maintain a signature of almost every malicious payload variant [18]. Behavioral signatures provide a more generic

description of the malware, making it more difficult to evade [18]. Unfortunately, behavioral signatures also suffer of the same limitations than payload signatures in that only known malicious behaviors can be detected. If a new trend of malware is developed with different behavior, the system will not be able to detect it [18].

In PFP, reference signatures are extracted from high-resolution readings from the power consumption during the execution of trusted software. Because any execution has an impact on the power consumption, it is extremely difficult for an intruder to hide malicious activity from our monitor. The main challenge in PFP is to properly characterize authentic software to yield these signatures.

Chapter 3

Power Fingerprinting: Theory of Operation

Power Fingerprinting (PFP) is a unique technique for execution integrity assessment on digital processors. PFP uses side channel information to detect deviations from the expected execution status. PFP is based on taking fine-grained measurements of the processor's power consumption and identifying the patterns that result from the specific sequence of bit transitions during execution. A signature extracted from the execution of trusted software is used as a reference to compare with test traces to determine if the same code is running. As mentioned before, PFP is similar to power-analysis side channel attacks used on smart cards and cryptographic devices. In both cases the power consumption during operation is used to determine execution details. The similarities between PFP and side-channel attacks end there. While side-channel attacks use detailed knowledge of the algorithms executed and a power consumption model to guess what parameters are used during the specific execution, PFP aims at characterizing the specific algorithm being executed and detect when that algorithm is being executed, and when it is not.

One interesting aspect of PFP is that it frames the tasks of integrity assessment and intrusion detection as a signal detection problem, for which there are rigorous analytical tools and

quantitative metrics already available. The existence of these analytical tools allows PFP to provide quantitative metrics of security and trust as well as to provide concrete statistical measurements of performance.

In this chapter, we describe in detail the operation of the proposed PFP integrity assessment monitor.

3.1 Software-Dependent Power Consumption

As mentioned in Section 2.1, digital CMOS circuits dissipate power in two modes: static and dynamic. The former is due to reverse bias leakage currents and is a function of the total number of transistors and the operating temperature. As the name implies, static power consumption remains constant independent of processor operation. Dynamic power dissipation has two main contributors: direct path currents and the charge and discharge of load capacitances [74]. Leakage and direct path currents have a relatively small contribution to the overall power consumption. The biggest contribution to the processor's power consumption is related to the charge and discharge of load capacitances all across the device. The load capacitance is made of parasitic (junctions, gates, etc.) and wiring (interconnections) capacitances that are charged and discharged every time a gate switches from 0 to 1 or 1 to 0, respectively.

In a CMOS digital processor, the dynamic power consumption is software-depended and is determined by the instructions being executed, including their parameters and addresses, as well as the inter-instruction transitions when the processor executes sequence of instructions. The dynamic power consumed in a specific clock cycle is determined by the total number of bit transitions that take place in that cycle. *In other words, the dynamic power consumption is related to the Hamming distance between consecutive execution states across all logic gates in the processor.*

Let x and x' be successive logic states in a processor while executing a target routine.

Then, the power consumption of the device at the time it switches from state x to x' is proportional to the Hamming distance, $H_D(x, x') = H_W(x \oplus x')$, where H_W is the Hamming weight, or the total number of logic ones, and $\oplus$ is the Exclusive OR operation [55]. The execution of a specific software routine, when going through a determined sequence of states, yields a specific power consumption pattern, or fingerprint. This power fingerprint may be buried in noise and hard to model but it exists and carries information about the execution environment. It is this side-channel information that the PFP monitor utilizes to identify execution deviations by detecting anomalies on the power consumption.

3.2 The Power Fingerprinting Monitor

A PFP integrity assessment monitor is an external device that reads the instantaneous current drain between the power supply and the main processing hardware, extracts discriminatory features from the captures traces, compares them against stored signatures, and determines whether the observed traces match the signatures or not. The structure of the proposed PFP monitor is depicted in Figure 3.1. It consists of three main elements common to all pattern recognition systems: sensing, feature extraction, and classification. The sensor takes a direct, or indirect, physical measurement of the instantaneous current drain. The signal produced by the sensor is digitized at a high speed by a device that can be embedded on the target board or separated, as implied on the picture by including a COTS oscilloscope. The high resolution traces are analyzed to extract the necessary discriminatory features which are compared against stored signatures by a detector, which makes the final decision of whether the traces match the stored signatures or not. The following sections provide further details on each one of the modules.

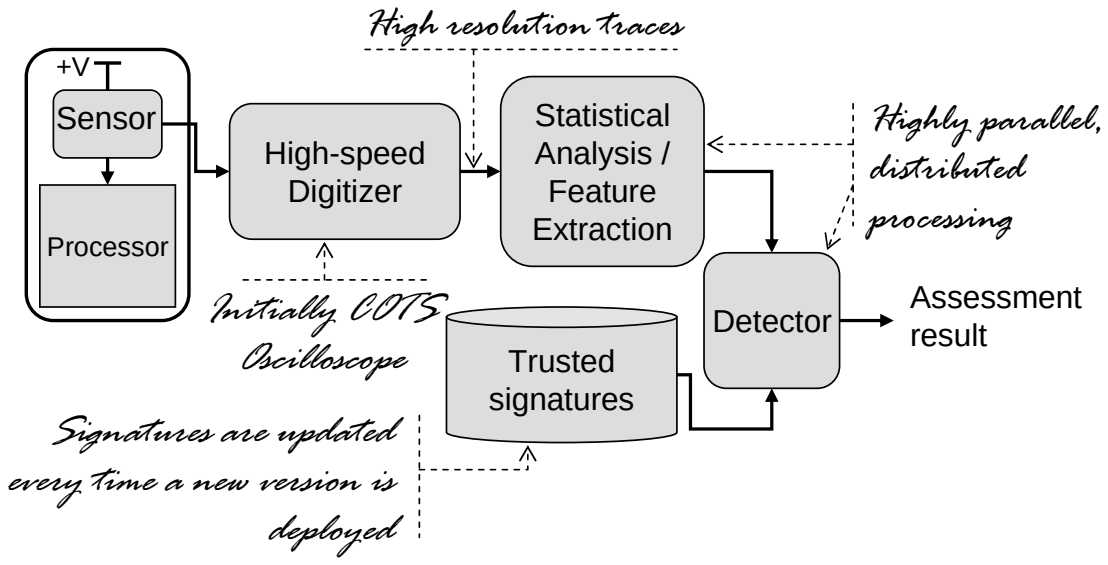

Figure 3.1: PFP monitoring system description

3.2.1 Sensing

Sensing is a critical aspect for the success of PFP. The sensor needs to collect a direct or indirect metric representation of the dynamic power consumption or instantaneous current drain of the processor. There are several technologies that can accomplish this task. For example, the sensor can be implemented by means of a commercial current probe, a Hall effect sensor [30], composite magnetic field sensor, Rogowski coil, a high-bandwidth modified Wilson current mirror [43][51], or a simple low-resistance precision shunt resistor in series with the core power supply. Notice that the sensor needs to meet the requirements set by the specific features extraction techniques selected and specific characteristics of the hardware, such as clock rate.

It is very important to mention that there are other side channels, besides power consumption, that carry execution status information from the processor [30, 55]. For example, temperature and processing timing can provide valuable information about the execution status. Electromagnetic radiation is another side channel that can provide similar signatures as the ones used in PFP. This relationship is evident when considering that the intermittent

high-intensity current drain responsible for dynamic power consumption also generates an electromagnetic field around the power rails and other interconnections.

The current sensor has to be physically located within the target platform, which must include provisions (contact points or pins) for this, but the actual sampler or digitizer can be an external device, such as a COTS oscilloscope. Once sampled, the high-resolution traces can be processed in place, transmitted to a different location, or stored for posterior processing. The physical location of the sensor is a critical element for the success of this approach. The ideal location, shown in Figure 3.2, right next to the power pins (or balls) of the processor, without decoupling capacitors between the sensor and the processor. If this location is not feasible, or introduces excessive power supply noise [1], then the second best location is right before the decoupling capacitors. This second location will receive attenuated traces, but it generally presents much easier access to place a sensor. When the sensor is placed in the second location the copper traces with their parasitic capacitance and inductance along with the decoupling capacitors create a low-pass (LP) RLC filter, with transfer function $H(f)$, that affects the current traces. The performance of PFP can significantly improve if $H(f)$ is pre-characterized using a Network Analyzer or another system identification technique [29]. The effect of the LP filter can be minimized by passing the traces through another filter with the inverse transfer function, $H^{-1}(f)$ or its closest stable approximation. There are more potential locations for the sensor, but the further away the sensor is placed from the power pins of the processor, the more potential interference is allowed and the more complex the transfer function becomes.

In Figure 3.2 VDD_core can be provided by different sources. For simple processors, such as 8-bit PICs, it comes directly from the voltage regulators. For more sophisticated platforms, such as modern ARM processors, it can come from a power and peripheral management system. This power management system is a complex circuit that provides a wide array of services including different voltage regulation levels, reset and interrupt handling, and

[1] Longer copper trace between the power pins and the decoupling capacitors will increase parasitic inductance

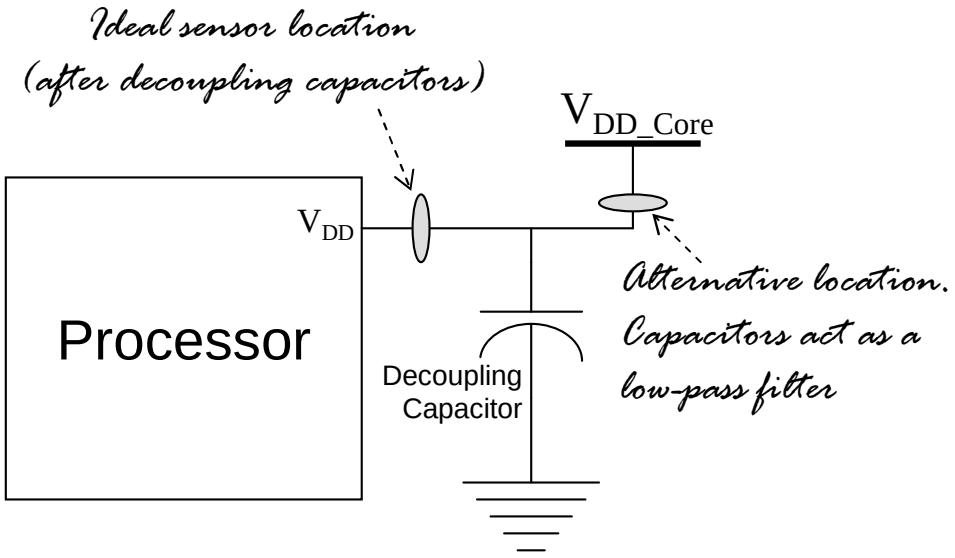

Figure 3.2: PFP sensor locations

other peripheral management. Power managers are complex systems that merge different signals and add interference from the PFP perspective and tend to hide the power signatures. For system with a power management circuit, it is recommended to design the system board with the necessary provisions to place the current sensor after the power management system to avoid the extra interference and facilitate signature extraction. In a best case scenario, the power sensor would be included into the power management system as another service provided, facilitating the integration of PFP.

In modern complex systems, there are different "power domains" in which several subsystems share the same voltage levels and start-up sequence. Subsystems within a power domain, use power from the same set of pins in the processor. For PFP, having multiple subsystems in the same power domain as the main core can be sees as a source of interference. There are some ways to avoid this interference by providing a separate power rail to the pins that are closer to the main core, but this requires deep knowledge of the internal construction of the system and a unorthodox design methodology. For this work, interference due to extra subsystems in the same power domain is treated simply as noise.

In the case of multiple processors in the board, the same principle can be repeated for each processor, as shown in Figure 3.3. In this case, the detector must be designed to combine and consider traces from both sensors. For multiple processor cores in the same package, the same principles apply as in the multi-processor case and it is necessary to provide a separate power rail to the power pins of each core.

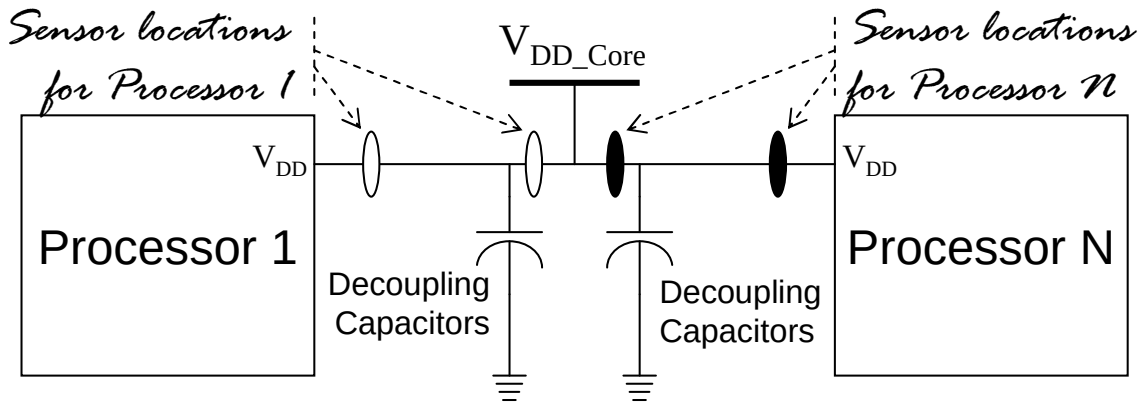

Figure 3.3: PFP sensor locations for multi-processor boards

3.2.2 Feature Extraction

One of the most important elements in PFP is the identification of those statistical and temporal properties of the power consumption (feature extraction) that uniquely identify the execution of a given software routine. In feature extraction the goal is to gather captured traces from the sensor, process them, and prepare them to be passed to the detector so it can make a decision of weather they match the stored signatures. Different software routines will likely require different feature extraction techniques in order to capture the unique characteristics that better define them. There are a multitude of different techniques that can be used to extract discriminatory features from power traces, from time domain correlation, power spectral density, higher-order statistics, cyclostationarity, wavelet decomposition, just to mention some. Unfortunately, there is no effective procedure to determine the optimal features, or what techniques will yield the best discriminatory qualities. Selecting discrimi-

natory features requires a great deal of heuristics and empirical results to identify the best feature extraction techniques.

Feature extraction in PFP is a challenging task that requires deep understanding of the processor's architecture and the software structure, but which can be facilitated by structuring the software so it displays specific characteristics. Basic software structures with deterministic execution are easier to characterize in PFP. Hence, systems with large content of signal processing code and relatively deterministic execution, such as SDR, are fitting candidates for PFP characterization. It is important to note that, due to the statistical nature of different discriminatory features, several executions cycles may need to be observed in order to provide adequate performance. Furthermore, discriminatory are not limited to a single domain, as a number of different techniques can be used to provide better discriminatory qualities and improve the assessment performance of PFP.

In some cases, the process for extracting discriminatory features can be quite complex. In this cases is convenient to separate the process in two: preprocessing and the actual feature extraction. Preprocessing involves general conditioning tasks prepare the traces and facilitate the rest of the process. Preprocessing examples include converting the traces to the appropriate domain, or aligning the traces in reference to a specific marker. A specific example of trace preprocessing is shown in Figure 3.4, in which time-domain traces from the execution of test software in an BeagleBoard with an OMAP3 processor are first converted to the frequency domain by calculating their power spectral density.

Feature extraction in PFP is the process of calculating the final test statistic which is passed to the detectors and used to determine integrity. The specific process is unique to each selected feature. Following our previous preprocessing example in which the PSD traces is extracted, the actual discriminatory feature can be considered to be difference, in dBs, between the PSD of a new trace and the stored signature, as shown in Figure 3.5. In this case, feature extraction would be the actual subtraction process. The resulting final test statistic is passed to the detector.

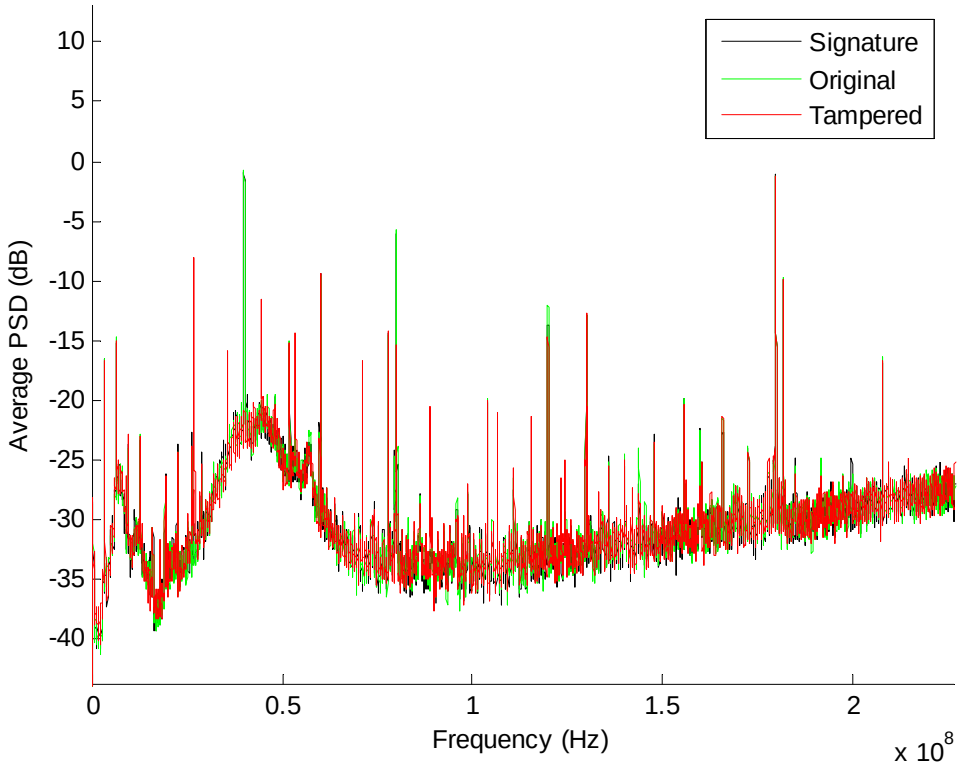

Figure 3.4: PFP feature extraction using power spectral density

In another basic example, preprocessing can consist on coarse synchronization by aligning time-domain traces and some compensation for environmental factors, as shown in Figure 3.6, while feature extraction involves comparing against the stored signature by calculating the correlation factor or the Euclidean distance using each sample in the captured time-domain trace as a dimension in an Euclidean space.

These are just examples to illustrate the feature extraction task. The approaches, however, are real and are used as the basis for some of the results shown in later chapters that include details on the specific feature extraction techniques used.

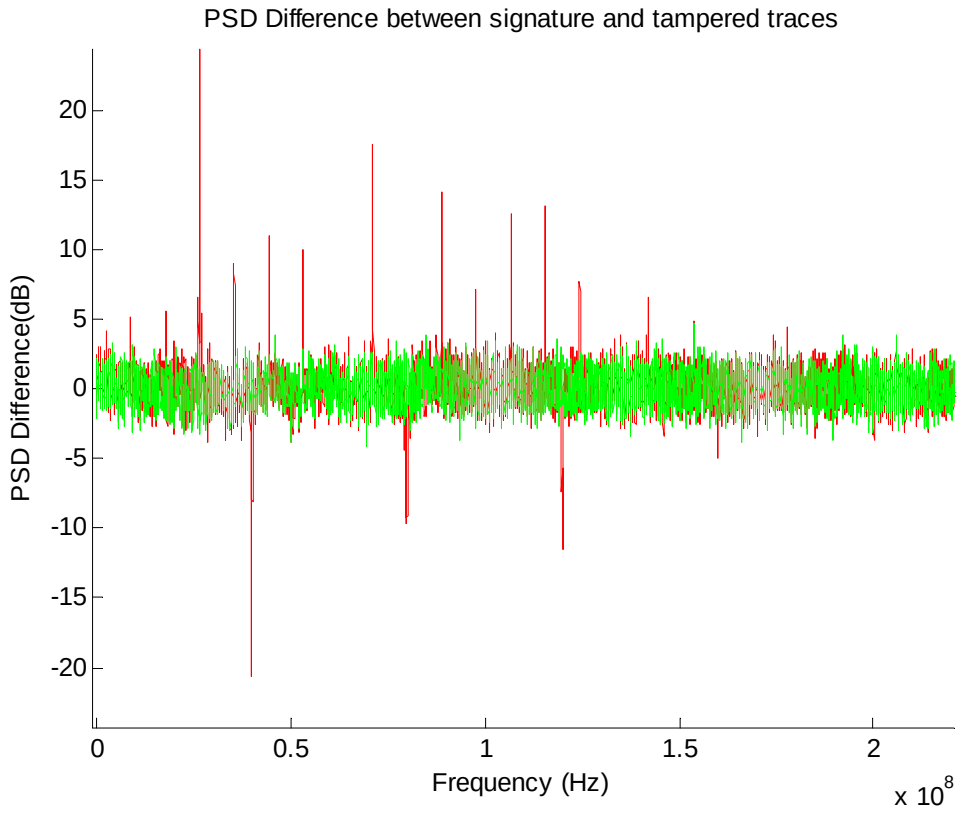

Figure 3.5: PFP feature extraction by calculating the PSD difference from test traces against a stored signature

3.2.3 Classification

As mentioned before, PFP can be seen as a signal detection and classification problem in which a carefully designed detector decides whether features from an observed power trace correspond to authorized execution. This task is an example of binary classification because the trace either matches or not. In PFP, the detector compares features from incoming power traces against all stored signatures. When the observed traces cannot be matched to any of the stored signatures, within a reasonable tolerance, it is determined that an intrusion has occurred. While the actual detection operation is relatively easy, the design process requires careful realization, as the performance of PFP depends on the accuracy of the detector at separating anomalies form normal operation.

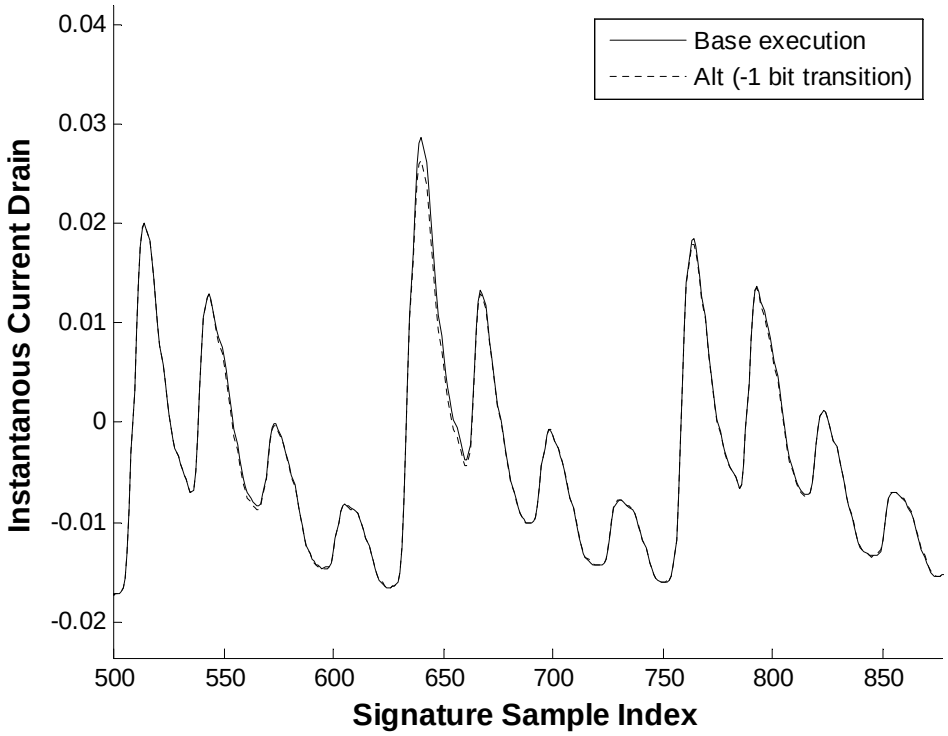

Figure 3.6: PFP feature extraction using time-domain traces

There are several detector architectures and the choice of detector depends on the specific features selected. For example, for clustering features, a basic approach is to implement a minimum-distance classifier, in which an unknown trace is labeled as a member of the class whose class prototype, or centroid, is the closest. For this case, a linear decision function would be effective. There are many factors that impact the location of the decision function (the linear discriminant or threshold) that determines when a trace is considered anomalous, including the statistical characteristics of the traces and the a-priori knowledge available about the statistical properties of the training traces probability distributions.

In PFP, all is needed for the detector to make a decision is a reference signature to compare the incoming traces against and a criterion to make a decision. These two elements are determined during a process called module characterization. In the next section we provide a more detailed description of this process.

3.3 Module Characterization

The dependency between software execution and power consumption in digital processors has been extensively documented. What makes PFP different from other approaches and turns it into an effective tool for integrity monitoring, is the reference signatures from the execution of trusted modules. These power signatures, or fingerprints, are what give PFP its power and effectiveness. The signatures are extracted in a process called module characterization, which is arguably the most critical, and difficult, aspect of PFP.

The ultimate objective in module characterization is to provide the PFP monitor with all the necessary information to assess the execution integrity of a given software module. There are two critical pieces of information:

- What the power traces are supposed to look like
- How much deviation from them can be considered normal

Both elements are extracted by analyzing a set of training traces from the execution of a trusted module. The former, called signature extraction, involves identifying the discriminatory features from the training traces that better distinguish them from other executions. The latter, called detector design, requires statistical analysis on training traces to identify at what point the a trace becomes an anomaly. The following sections provide more details on these critical tasks. The characterization process is depicted in Figure 3.7.

3.3.1 Signature Extraction

Signature extraction relies on capturing training traces, which is accomplished by repeatedly executing the trusted target code in a controlled environment while collecting the power traces during its execution (using the same sensor/digitizer used in PFP). Because PFP is expected to be an extremely sensitive monitoring approach, it is important that the exact

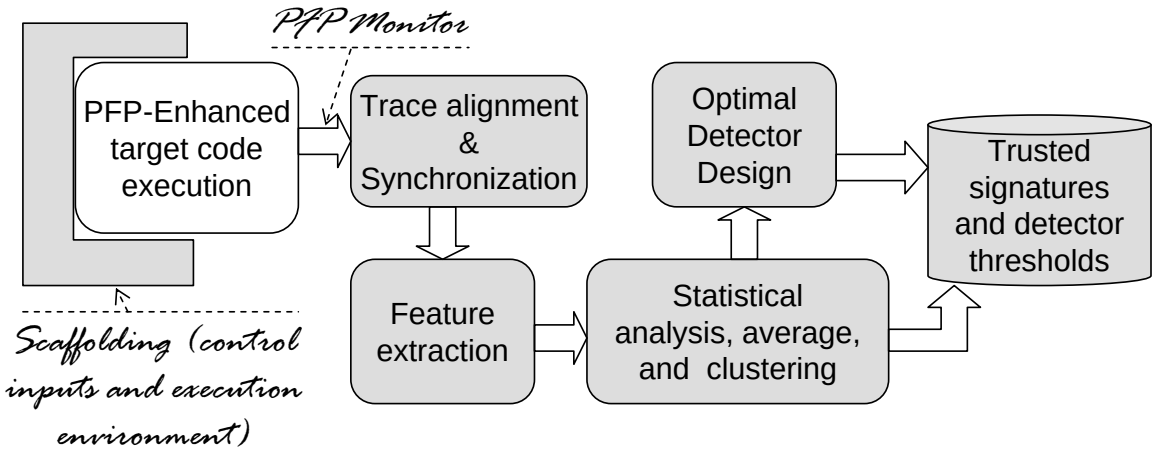

Figure 3.7: Trusted code characterization process

code that will be deployed be used during characterization; this includes using the exact same tools to build the software, with the same level of optimization, etc. The general idea is to repeatedly execute the target code in the controlled environment, provided by custom made scaffolding software (similar to the one used for automatic unit testing), while observing its power consumption which is analyzed to determine the features that best describe it.

The signature extraction process can be divided in two phases: trace collection during controlled execution and feature extraction. The elements controlled by the scaffolding software include: isolating the target software, setting inputs used during execution, and inserting specific markers that help synchronizing traces. Feature extraction, as mentioned before, relies on heuristics and designer experience to determine the features that will deliver the best results. Features can be extracted from different signal domains and be multidimensional. Furthermore, multiple signatures can be used to identify a single piece of code.

Ideally, a signature is extracted from every software execution path, as control structures such as "if" statements can yield different signatures. In practice, this is not feasible even for moderately complex systems. In these cases, only a few critical modules are characterized and monitored, such as OS kernel, configuration, or cryptographic modules. This is the approach used for the results presented later in this document.

For better performance, the characterization should be an iterative, interdependent process. During which the source code structure along with the respective markers are co-developed to yield the strongest signatures with the smallest variance across different execution instances. Several traces from the execution of the trusted code may need to be collected in order to average them and reduce the impact of random noise inherent to any physical system.

3.3.2 Detector Design

Detector design is the final step in the characterization of a target module. The objective of this task is to perform statistical analysis on the features extracted from the training traces to determine a threshold under which an observed trace can be considered, with a high level of confidence, to have originated by the untampered target software.

Due to inherent noise in the captured traces, features extracted from different execution instances of the same software modules will display a certain level of variance. An example of this distribution is shown in Figure 3.8. An adequate threshold needs to be identified based on the statistical features of the training traces.

Detector design has its foundation in the fields of pattern recognition and statistical learning, from where several algorithms and approaches have been developed to design detectors and classifiers. The specific choice of detector design depends on several factors including: feature selection, available resources, a-priori knowledge of the system, tolerance to errors, etc. One of the simplest approaches to pattern classification involves clustering approaches based on distance functions.

Minimum Distance Classification and Clustering

Consider a pattern vector (discriminatory features from a power trace) $\mathbf{x} = \{x_0, x_1, \ldots, x_{n-1}\}^T$. This vector represents a point in an n-dimensional feature space. The Euclidean distance

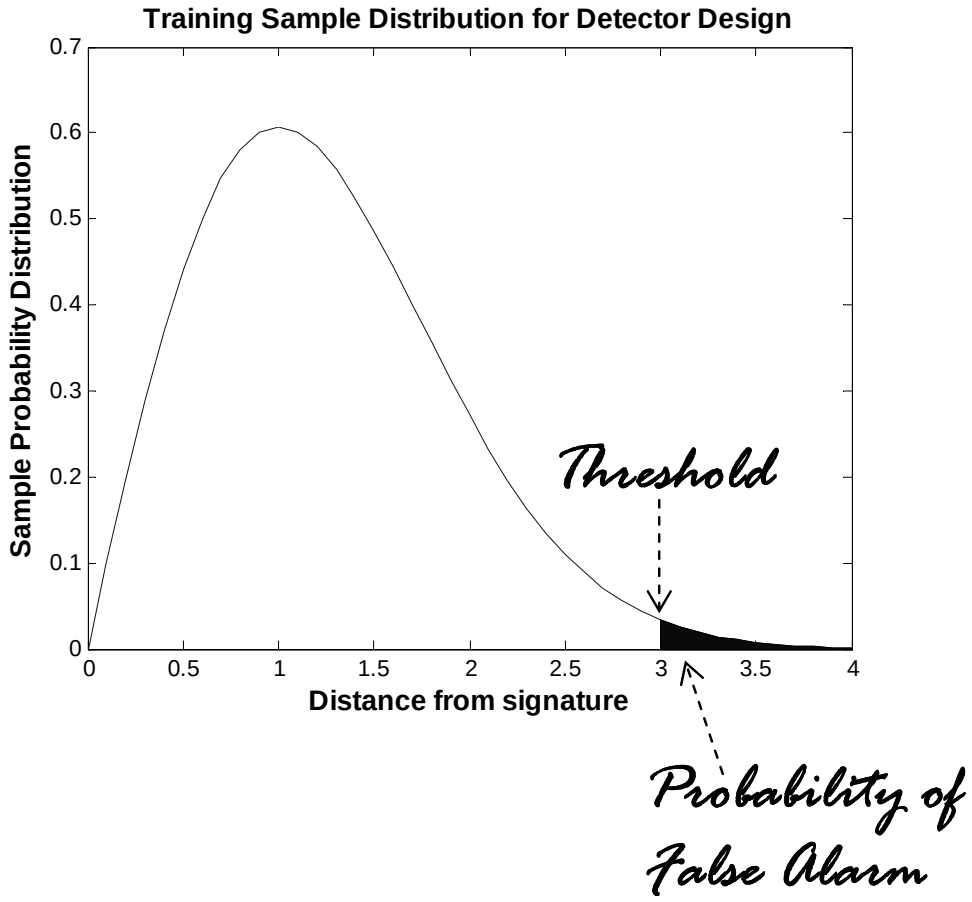

Figure 3.8: Sample probability distribution from trusted code execution used for PFP detector design and threshold selection

between two vectors is given by

$$D_{\mathbf{xy}} = \|\mathbf{x} - \mathbf{y}\| = \sqrt{(\mathbf{x} - \mathbf{y})'(\mathbf{x} - \mathbf{y})}$$

It is expected that patterns produced by class[2] i will form a cluster around a specific center, or class prototype $\mathbf{z}_i$. Consider a system that needs to classify a pattern $\mathbf{x}$ of unknown classification into one of M pattern classes with prototypes $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_M$. The distance between $\mathbf{x}$ and the ith prototype is represented by D_i. Hence, a straight forward approach to assign $\mathbf{x}$ to a class, is to calculate the distance to all the different prototypes and select

[2]A class represents all the power traces generated by the execution of a given target software

the one that is closest. In other words, $\mathbf{x}$ is assigned to class w_i if $D_i < D_j$ for all $j \neq i$. Such classifier is known as a minimum-distance classifier.

Likelihood Function Classifiers

In this section we provide a brief introduction to likelihood function classifiers and different criteria used to make optimal classification decisions based on some specific metric of "goodness". We start by introducing some basic concepts from hypothesis testing which bring powerful tools for decision making and statistical pattern classification.

Assume that we have a power trace consisting of a single sample, x. This is an over simplification of a real system, but it is used only to introduce some basic concepts. We want to classify this test trace as being the generated by the execution of untampered trusted software or some other unauthorized software. There are two hypotheses:

$$\begin{aligned} H_0 &: \text{ Authorized execution. No tampering} \\ H_1 &: \text{ Unauthorized execution. Tampering} \end{aligned}$$

We want to determine what hypothesis is most likely to be true based on the test trace x. We have two conditional probabilities

$$P\{H_0|x\} \quad \text{and} \quad P\{H_1|x\}$$

which represent the probability that either H_0 or H_1 is true given that we observed trace x. A reasonable criterion based on these *a posteriori* probabilities would be to choose H_0 if

$$P\{H_0|x\} > P\{H_1|x\} \quad \text{or} \quad \frac{P\{H_0|x\}}{P\{H_1|x\}} > 1$$

and choose H_1 otherwise. We can describe this in terms of probability distributions as

$$P\{H_0|x \leq X \leq x + dx\} > P\{H_1|x \leq X \leq x + dx\}$$

Applying Bayes's rule we can express

$$P\{H_0|x \leq X \leq x + dx\} = \frac{P\{H_0\}P\{x \leq X \leq x + dx|H_0\}}{P\{x \leq X \leq x + dx\}} \tag{3.1}$$

where $P\{H_0\}$ is the *a priori* probability that H_0 is true. If we assume x has a probability distribution function $p(x)$ we have

$$\begin{aligned} P\{x \leq X \leq x + dx\} &= p(x)dx \quad \text{and} \\ P\{x \leq X \leq x + dx | H_0\} &= p_0(x)dx \end{aligned}$$

where p_0 is the *a posteriori* probability distribution given that H_0 is true. Using these expressions we can rewrite Equation 3.1 as

$$P\{H_0 | x \leq X \leq x + dx\} = \frac{P\{H_0\}p_0\{x\}dx}{p\{x\}dx}$$

In the limit, as dx goes to 0 we have

$$P\{H_0|x\} = \frac{P\{H_0\}p_0\{x\}}{p\{x\}}$$

We can also follow a similar procedure for H_1, using $P\{H_1\} = 1 - P\{H_0\}$

$$P\{H_1|x\} = \frac{[1 - P\{H_0\}]p_1\{x\}}{p\{x\}}$$

Hence, the maximum *a posteriori* criterion is now to choose H_0 when

$$\frac{P\{H_0\}p_0\{x\}}{[1 - P\{H_0\}]\, p_1\{x\}} > 1 \quad \text{or} \quad \frac{p_0\{x\}}{p_1\{x\}} > \frac{1 - P\{H_0\}}{P\{H_0\}}$$

The ratio $\frac{p_0\{x\}}{p_1\{x\}}$ is known as the likelihood ratio. $p_0\{x\}$ and $p_1\{x\}$ are the likelihood functions and they represent the probability distributions of the captured trace x given that H_0, or H_1, are true.

It is possible to make two kinds of errors in this scenario: Decide H_1 when H_0 is true, described as $D_1|H_0$ or *false alarm*, and the opposite, decide H_0 when H_1 is true, described as $D_0|H_1$ or *missed detection*.

Bayes Classifier

The previous analysis provides a decision criterion optimal in terms of *a posteriori* probabilities. There is no sense of penalty incurred when making a mistake. If we define C_{ij} as the

cost of deciding D_i when H_j is true and assume that the cost of making an error is bigger than the cost for making a correct decision, or in our case

$$C_{01} - C_{11} > 0 \quad \text{and} \quad C_{10} - C_{00} > 0$$

then, the average cost is given by

$$\begin{aligned} \bar{C} = \quad & P\{H_0\}\left[C_{00}P\{D_0|H_0\} + C_{10}P\{D_1|H_0\}\right] + \\ & \left[1 - P\{H_0\}\right]\left[C_{01}P\{D_0|H_1\} + C_{11}P\{D_1|H_1\}\right] \end{aligned}$$

The Bayes criterion, which minimizes the average cost, can be shown [75] to yield the decision rule to choose H_0 when

$$\frac{p_0\{x\}}{p_1\{x\}} > \frac{1 - P\{H_0\}(C_{01} - C_{11})}{P\{H_0\}(C_{10} - C_{00})} \tag{3.2}$$

Notice that when $C_{01} - C_{11} = C_{10} - C_{00}$ this criterion is equivalent to the maximum *a posteriori* criteria described above

Neyman-Pearson Criterion

In power fingerprinting, the *a priori* probabilities $P\{H_i\}$ cannot be calculated accurately, since we do not know the statistical characteristics of tampering. It is also difficult to determine an adequate cost for making a decision error. In this case, similar to radar, it is adequate to apply the Neyman-Pearson criterion which maximizes the probability of detection for a given probability of false alarm. The Neyman-Person criterion states that there exists a non-negative number η such that if hypothesis H_1 is chosen when

$$\frac{p_1(x)}{p_0(x)} \leq \eta$$

and H_0 is chosen otherwise, then this rule yields the maximum $P\{D_1|H_1\}$ for all patterns subject to the constraint that $P\{D_1|H_1\}$ is less than some predetermined constant α. It can be shown that maximizing $P\{D_1|H_1\}$ is equivalent to minimizing

$$Q = P\{D_0|H_1\} + \mu P\{D_1|H_0\}$$

where μ is an arbitrary constant. Notice that this is a special case of the Bayes criterion. Substituting $C_{00} = C_{11} = 0$, $[1 - P\{H_0\}]\, C_{01} = 1$, and $P\{H_0\}C_{10} = \mu$ in Equation 3.2 we have the average cost $\bar{C} = Q$ which has been already shown to be minimized. Therefore, the decision rule is to choose H_1 when

$$\frac{p_1(x)}{p_0(x)} \geq \mu$$

where μ is chosen to satisfy the false alarm probability constrain.

Detector Design in PFP

In detector design, test traces from the execution of trusted software are captured and processed to extract the selected discriminatory features and compared against the stored signatures. Several traces are collected and processed and their statistical sample distributions are used to identify a threshold that yields the expected performance targets. The process of detector design is shown in Figure 3.9.

A common approach to create optimal detectors involves the application of the Neyman-Pearson criterion to maximize the probability of detection for a given probability of false alarm. It is important to note, however, that there are different techniques that can yield improved results depending on the nature of the selected discriminatory features. Other techniques for detector design and machine training include: Neural Networks, Support Vector Machines, and Hidden Markov Models.

3.4 Synchronization

Both aspects of PFP, module characterization and operational integrity assessment, depend on analyzing the traces that correspond exactly to the execution of the target software. In other words, the PFP monitor needs to be synchronized with the software execution. Synchronizing an external device with the internal execution status of the processor is a difficult

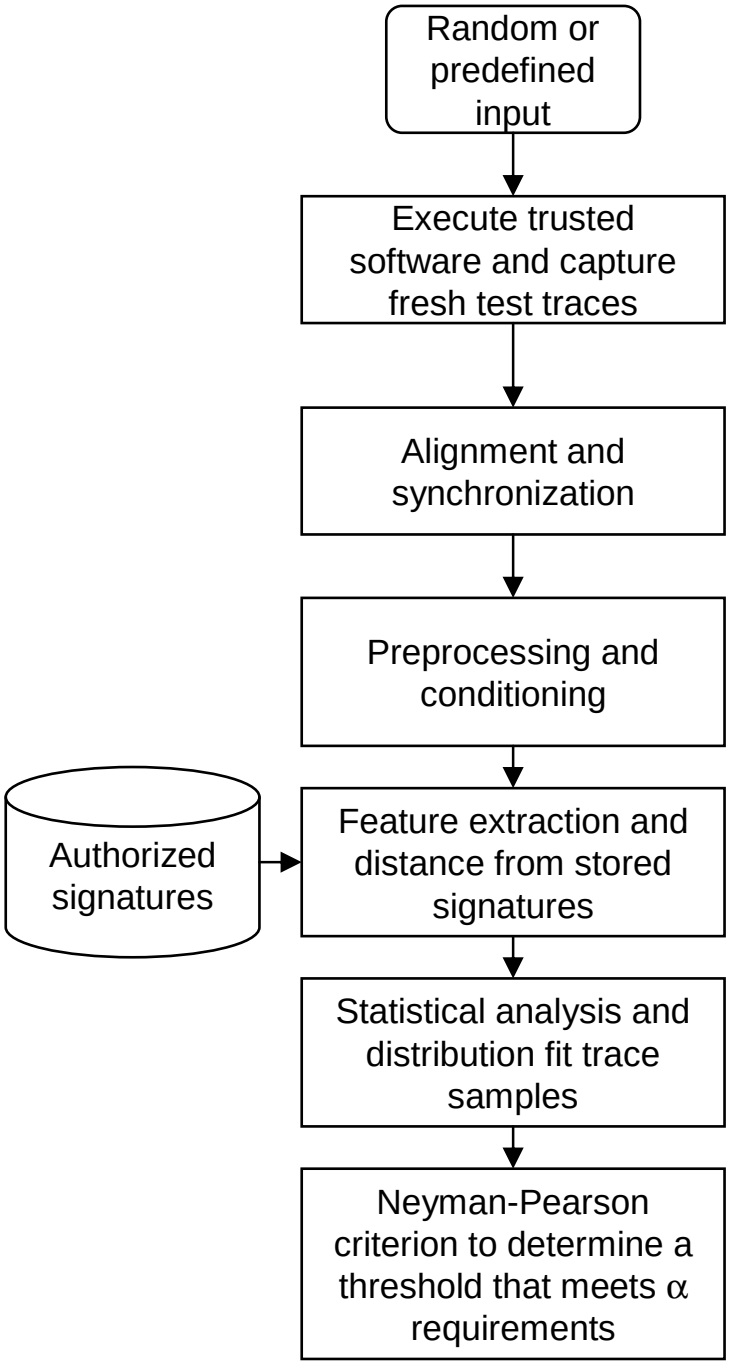

Figure 3.9: Flow diagram of the detector design process in PFP

task because it requires some form of signaling between the processor and the monitor. As expected, synchronization between the execution and the PFP monitor is greatly simplified if the target software already provides the necessary signaling. This signaling should not affect the application's logic, but it will necessarily add extra cycles. **This extra processing necessary for signaling is the only overhead that PFP monitoring may introduce to a target system.**

Signaling overhead is not always necessary. There are certain cases in which it is possible to simply execute the target module in an infinite loop during characterization and use spectral analysis to extract the necessary timing information and avoid introducing signaling overhead. This approach is effective for static software structures running on deterministic platforms.

For the cases in which PFP synchronization signaling is necessary, there are different options available. We consider two potential options: physical signals (as changing the voltage level of a pin) or a specific sequence of instructions that yields a known power consumption sequence. An example of a physical trigger signal is shown in Figure 3.10. Physical signaling is a simple approach, in which a specific value is written to designated IO pins in the processor. The PFP monitor simply adds an extra connection to read the value of the trigger and capture power traces at the right time. In the example, the signaling is provided by using an existing LED port on the sample platform. The application designer simply needs to turn on and off the LED to signal the PFP monitor the execution of target software, usually right before it runs. All the monitor needs to do is to trigger trace capture depending on the status of the LED. This can be done by having an optical sensor that reads the LED status or by attaching a separate voltage probe to the processor pins.

Physical signaling is the simplest signaling approach, but it has its drawbacks. The most important one is that, by having a physical interaction, especially if the monitor reads the value of a processor's register, it violates the principle of separation. Therefore, it is technically possible to have a processor compromise affecting the monitor as well. Another

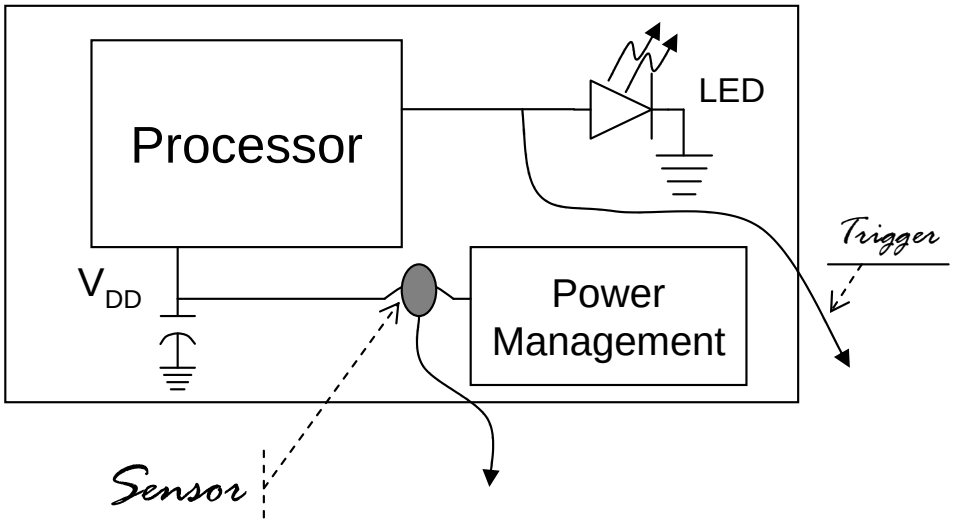

Figure 3.10: Example of triggering with a physical signal

important disadvantage is that, by having the synchronization routines within the processor scope, they fall within the range of cyber attackers. In other words, a malicious attacker, with knowledge of the PFP operation, can try to disrupt the synchronization signaling to avoid being detected.

It is also necessary to consider the indirect access to peripherals provided by certain processors and operating environments. For example, when the target application is running on the User Space in a Linux platform, or in any other operating system with indirect access to physical signals, as described in Figure 3.11, it is necessary to account for the inherent uncertainties in execution and timing caused by the indirect access. This is because file access requires the process to wait (block execution) for the appropriate synchronization signaling during which the kernel schedules other process to run.

The other signaling approach considered in this work requires inserting a short sequence of instructions especially designed to produce a known pattern in the processor's power consumption. This approach is depicted in Figure 3.12. Executing this short signaling sequence is similar to transmitting a preamble training sequence in wireless communication systems. Usually, the preamble sequence is implemented by selecting instructions that yield

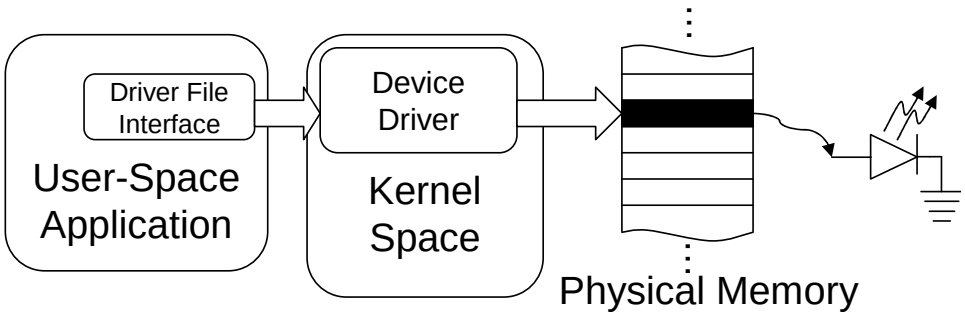

Figure 3.11: Indirect access to physical resources in the Linux device driver paradigm

a strong variation in the current drain. In addition, the sequence is usually repeated a few times to provide strong autocorrelation properties and facilitate preamble detection.

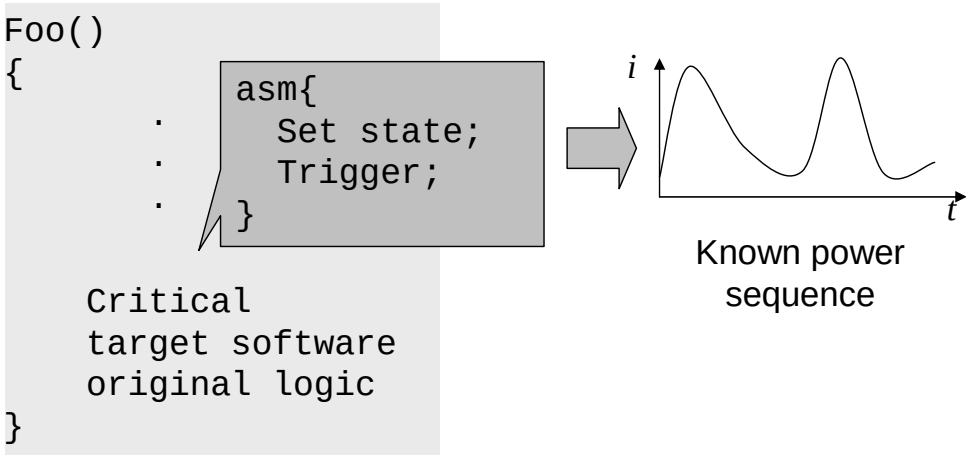

Figure 3.12: PFP strategic instruction insertion for synchronization and triggering

Embedding a PFP preamble has the extra advantage of reducing the impact of deeper pipelines in modern processors by providing a buffer between the target software and the interference due to surrounding software. Executing a preamble routine loads the pipeline with predetermined set of values before the target code executes.

Embedding synchronization signaling in the power consumption of the software has the advantage that it does not require extra circuitry as is needed to read the physical signals.

On the other hand, creating a preamble sequence with good detection properties is not trivial and requires deep understanding of the processor's architecture, as the instructions need to be selected such that the operational codes, addresses, parameters combine at the right time in the processor pipeline to produce the required results.

Similar to physical signaling, embedding a synchronization preamble in the power consumption has some drawbacks. For instance, it requires extra design time and the processing overhead can be more significant if the sequence of instructions necessary to provide good detection properties is too long. In addition, it also makes the system vulnerable to a knowledgeable attacker emulating a preamble. It is important to mention that a system with embedded PFP preamble is more robust to attacks to the monitor than a system using physical signaling, as the preamble is more difficult to disrupt.

Under certain circumstances, it is possible to remove the signaling artifacts once the modules have been characterized. Currently, this is limited to deterministic platforms in which removing the signaling does not affect the location in memory of the target software (for physical signaling, this can be accomplished by signaling *after* the execution of the target software). The run-time assessment process, however, is facilitated if the signaling artifacts remain in place. In the case when the markers are left on the deployed version, it is necessary to ensure that the facilities or services used for the markers will still remain in the deployed platform (e.g. if the marker is supposed to turn on a LED, that LED must exist on the deployed platform). For embedded PFP signaling, the preamble must remain in the final deployed system, or careful measures must be taken to prevent impacting the resulting power signatures.

3.5 Monitoring Operation

Once signatures have been extracted from the execution of trusted code, discriminatory features have been selected, and optimal detectors have been designed, the PFP monitor is

ready to assess the integrity of target software. As mentioned before, the normal integrity assessment process, depicted in Figure 3.13, is very similar to the detector design process.

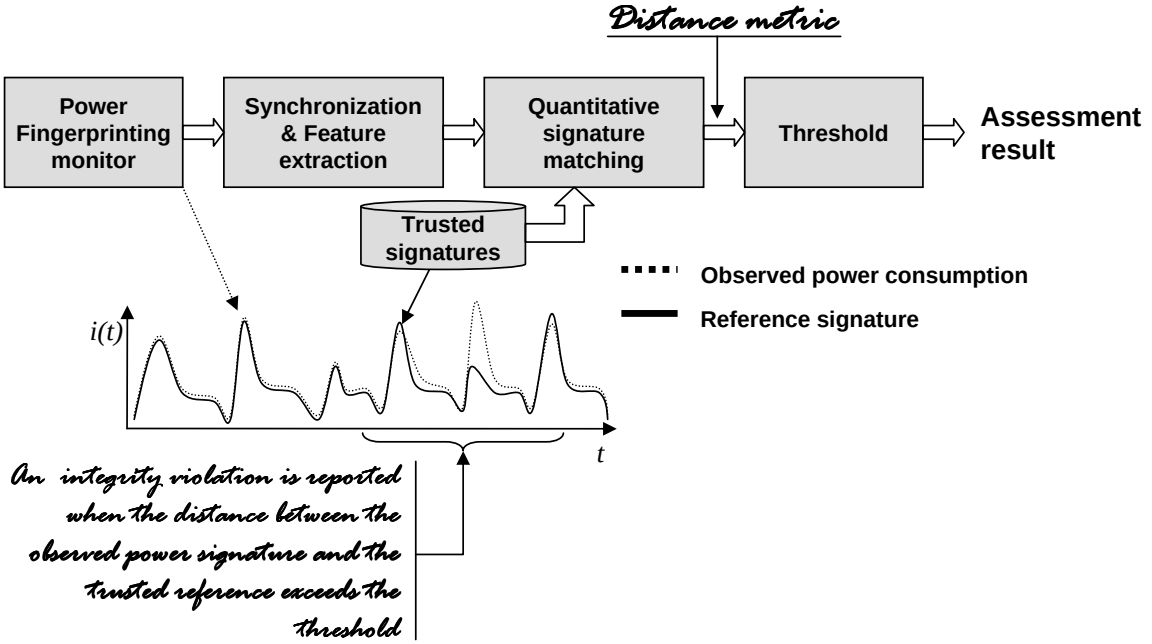

Figure 3.13: PFP integrity assessment operation description

During normal operation, the monitor extracts the selected discriminatory features from power traces after the necessary preprocessing, but instead of collecting the statistics from several traces, as was done for detector design, they are passed immediately to the appropriate detector to compare against the respective signatures and thresholds to determine the integrity status of the test code execution. The detector compares the test traces against all known signatures and, if no single test statistic is enough to determine that authorized code has executed, then an intrusion is reported. Keep in mind that power signatures must be extracted from trusted software before it is deployed and they need to be updated with every software revision.

3.6 PFP Applications

Effectively, PFP allows an external device to observe the internal execution status of a processor, allowing external execution monitoring with little to no cooperation from the target processor. PFP provides an extremely accurate mechanism to determine whether a given execution corresponds to a pre-characterized module. Operationally, PFP provides a "tamper seal" that indicates when execution violations have taken place. It provides a reliable trust indicator, accurate integrity assessment, and early warning (almost immediate) in the event of tampering.

Integrity assessment has a direct application in several fields, but we concentrate on two specific areas critical for SDR: 1) regulatory compliance, to detect when a device that has been certified to operate with a specific hardware-software pair is illegally modified; and 2) security, to detect malicious intrusions.

For SDR regulatory compliance, the fingerprint of the authorized software is stored during device certification and then used at runtime to verify that only the software used during certification is executed. Regulatory entities can leverage their current mechanisms with this approach to ensure that critical modules that impact spectral emissions are not modified after deployment.

For security, PFP has a clear application in intrusion detection. For instance PFP can detect a violation the precise moment a privilege escalation attack manages to break the existing containment mechanisms.

There are other application areas in which PFP can result beneficial, including timing property evaluation, post-mortem analysis, digital-rights management, etc. but a description of applying PFP to these fields is out of the scope of this work. In the next chapters we will provide copious evidence of the feasibility of applying PFP in the fields of security and SDR regulatory compliance.

3.7 Advantages of Power Fingerprinting

PFP allows an external monitor to peek inside the processor and extract execution status information. There are clear advantages that come with having such a monitor. For instance, because PFP uses fine-grained current sensing, it captures the bit transitions that happen every instruction cycle. This high-resolution sensing provides the monitor with significant visibility into the internal execution status, making it extremely difficult for attackers to evade the PFP monitor, as they would have to match perfectly the power consumption of specific code. Furthermore, because the monitor is a device physically separated and completely independent from the processor, it provides effective isolation to prevent attacks on the monitor itself. This isolation, however, is achieved without compromising visibility due to the use of the power consumption side channel.

The signatures in power fingerprinting are taken from the execution of authorized code. This represents an alternative to traditional signature-based intrusion detection techniques which extract signatures, functional or behavioral [18], from the malware itself. These traditional monitors are susceptible to evasion by simple polymorphism and obfuscation and are ineffective against malware for which there is no signature available. One more important advantage of using power fingerprinting to monitor execution status is that it can reliably detect intrusions regardless of the attack model. If the attacker exploits a memory corruption vulnerability (such as a buffer overflow), or is a malicious insider with privileged access, a power fingerprinting monitor still has a chance of detect the unauthorized insertion.

From a regulatory perspective, PFP creates a tie between hardware and a specific software version, as the fingerprints depend on the specific power consumption characteristics (determined by hardware) and the specific sequence of bit transitions (determined by software). Therefore, PFP provides regulatory bodies and equipment manufacturers with a mechanism to detect when devices are being operated with illegally modified or unauthorized software versions.

The main advantages of power fingerprinting are summarized on Table 3.1.

Table 3.1: Advantages of PFP intrusion detection

Enabling principle	**Resulting advantage**
Improved visibility using fine-grained power consumption monitoring	Improved resistance against evasion.
Physical isolation due to external monitoring	Increased resistance against attacks to the monitor itself
Minimum overhead and negligible impact on latency and processing load	Coverage of embedded, resource constraint, and legacy platforms. Stealth monitoring
Positive characterization and trusted signatures	Effective against zero-day attacks
Sound analytical framework	Quantitative metrics for security and trust. Probability of detection and false alarm

3.8 Limitations and Challenges

PFP provides a powerful monitoring tool for integrity assessment. The approach, however, only improves monitoring capabilities and needs to be deployed in association with traditional security techniques, such as tamper protection, to prevent attackers from gaining access to the system in the first place. Due to the statistical nature of different discriminatory features in PFP and noise inherent to power lines, several execution cycles may need to be observed in order to provide a reliable estimate of the execution status. The detection performance of this approach can be improved by integrating traces from several execution cycles or by adding observation points on the power rail or platform.

While the feasibility of PFP has been demonstrated and evidence is provided in the next chapters, there are still several challenges and risk areas that need to be addressed to provide a practical solution:

Interference & noise - In addition to the unavoidable Gaussian noise in captured traces, there are extra sources of variations that can affect the performance of PFP. Manufacturing variability and environmental changes, such as temperature and aging, affect the power consumption characteristics of the processor. While the changes are small and happen slowly, they raise noise levels on the traces and reduce the overall performance of PFP. These issues might be addressed by placing an adaptive filter, such as an equalizer, before feature extraction to compensate for the changes.

Interference caused by power consumption not related to the execution of software can be introduced by different components, such as peripherals, that are located on the same board. In this case, correct sensor placement can help eliminate most of this interference. In certain cases, however, interference will find its way to the PFP traces. For example, complex system that have separate peripherals in the same power domain as the processor core, or systems with multi-core processors and no room for multiple sensors, will have to deal with increased interference. In general, the performance of PFP improves when there are multiple sensors and critical sections are isolated, but that may require changes in the processor silicon and is outside the scope of this effort, which only targets single-core platforms.

Signature breakage - Multi-tasking OS and complex processor architectures can disrupt features and reduce performance. Depending on the nature of the target software being monitored, it can be interrupted by tasks of higher priority. When the OS preempts the execution of a target routine to schedule a different task with higher priority, the power signature will be disrupted. There are different ways to deal with this behavior. The first option is to insure that the target code is not pre-emptied by disabling interruptions in the system and giving it the highest priority. This solution guarantees that the signatures will not be broken, but also can add significant latency to the service of certain events. Another way

to deal with signature breakage is to characterize the context-switching operation (usually an interrupt service routine) to let the PFP monitor know when the task has been preempted and then piece the signature back together when the larger routine resumes execution.

Uncertainty on the traces - Random and human-dependent inputs introduce inherent variation on the traces. Optimal feature selection can be applied to remove the effect of the inputs and concentrate on the sections of the traces that correspond to instruction fetching and load as well as the addresses from the parameters. Under specific circumstances, it might be necessary to perform the integrity assessment using a predefined input to control the state sequence in the target modules and facilitate feature extraction.

3.9 Conclusions

This chapter described the structure and operation of a PFP monitor. The monitor includes a physical sensor that captures the instantaneous current drain of the target processor during execution and a high-speed digitizer to deliver high-resolution traces. Signal processing is performed on the traces to extract the discriminatory features that uniquely identify the execution of software. A detector loaded with pre-characterized signatures makes the final decision of whether the observed traces match the signatures or tampering has occurred.

One of the most critical aspects of PFP is the characterization of the target modules and software. During this process, reference signatures are identified and the specific thresholds to determine a match are calculated. There is a variety of techniques to perform feature extraction and the discriminatory qualities depend on the specific system being monitored. Feature extraction can be as simple as Euclidean distance on trace time sequence, or more complex analysis, such as wavelet decomposition. Unfortunately, there is no effective behavior to identify optimal discriminatory features and heuristic analysis is required. The selection of detector design depends on the specific features selected and can include minimum-distance classifiers, neural networks, or support vector machines, just to name a few.

Chapter 4

PFP in deterministic platforms

The operation and performance of PFP in intrinsically related to the hardware platform being monitored. There is a huge variety in processors and computing platforms, which spans a wide spectrum of capabilities, architectures, speeds, etc. The operation, tools, and approaches necessary to monitor a given platforms depend on the specific characteristic of the platform itself, and will be different when the target is a basic microcontroller than those necessary to monitor more complex platforms. Because PFP's reliance on anomaly detection, a more deterministic platform translates into a simplified signature extraction procedure and more reliable monitor performance.

This chapter describes the general characteristics of deterministic platforms and presents the application of PFP on them. It also provides feasibility results that demonstrate the viability of PFP in an embedded radio platform in two different application areas: security and regulatory certification. This chapter also describes specific techniques to improve the performance of PFP monitoring by precharacterizing the power consumption characteristics of deterministic platforms. This technique is used to determine the sensitivity of a PFP monitor to the smallest change from the software-related power consumption perspective. At the end, this chapter also describes mechanisms to handle intrinsic execution uncertainties due to operation with random parameters in deterministic platforms.

4.1 Deterministic Computing Platforms

Within the PFP context, a deterministic platform refers to a basic processor, often considered embedded, that takes the same steps when executing the same instruction, independent of the processor's state or logic around it. In other words, we consider deterministic platforms to be those that have a simple architecture and lack some of the features of more sophisticated processors, such as dynamic branch prediction [26], that make the operation of the processor dependent on the specific state sequence [1]. For example, we consider a deterministic platform to be a processor that when executing a given instruction (e.g. a NOP) the instruction always takes the same number of clock cycles to complete, independent of the state of the processor and assuming no interrupts happen.

This definition of deterministic platform is not universal and is not expected to be used outside of the context of PFP. There are several examples of deterministic processor. Most of what is known as an embedded processors or microcontrollers fall in this category. Some examples of popular platforms from the PFP perspective are shown in Table 4.1.

Table 4.1: Examples of deterministic platforms from the PFP perspective

Popular commercial deterministic platforms
Microchip's PIC family [46]
Texas Instrument's MSP430 family [69]
Atmel's AVR family [14]
Freescale's 68HC08/12 families [64]

A consistent trend within deterministic platforms is that they are often found in low-performance, low-power applications. Not surprisingly, they are also extensively used for highly-optimized, time-constraint applications, such as critical control and communication

[1] Notice that this is different from the potential effects of compiler optimization, that may remove irrelevant or unnecessary instructions

systems and signal processing.

4.2 General PFP Operation on Deterministic Platforms

From the PFP perspective, a deterministic platform presents a static target from which to extract baseline signatures that fully characterize selected execution paths. As a result, a PFP monitor can operate with confidence that no deviation from the execution is expected as a result from the hardware architecture itself. In other words, monitoring or characterizing a deterministic platform provides the assurance that any change detected in the power consumption, aside from noise and other interference sources, is a direct result from a deviation in the execution status and not a result of intrinsic execution uncertainty.

There is an important source of uncertainty in the power signatures, even in deterministic platforms: random parameters as inputs. While changes in random parameters do in fact alter the power signatures, their effects are limited to determined sections in the traces. Section 4.5, at the end of this chapter, describes a mechanism to deal with the effects of random parameters in deterministic platforms.

From a PFP perspective, a deterministic platform enables a characterization process in which it is only necessary to account for white noise in the measurements and any other source of interference that makes it into the traces, such as peripheral circuits and other board elements. This characteristic enables the extraction of basic discriminatory features, such as correlation, in the time-domain.

4.2.1 Early PFP results

The feasibility of detecting the execution of a precharacterized routine in using PFP was demonstrated in [1] for a basic deterministic 8-bit microcontroller. These results, however, come from a controlled execution environment running a carefully selected routine designed

to yield strong PFP signatures and not to accomplish a specific task.

One of these early results is reproduced here for the reader's convenience[2]. In this experiment, the target platform is a FOX11 Trainer board [77] which contains a Motorola's HC11 processor (68HC11E1), running at 2 MHz, with 32 KB RAM, 32 KB EPROM, and 32 KB EEPROM. The power traces are taken by measuring the voltage differential across a 1 Ω resistor in series with the main power supply with a Tektronix' TDS 694C digital real-time oscilloscope. The oscilloscope is configured to 500 MS/s and a capture length of 30000 points. The captured trace is transfered to the host computer using GPIB and all the post-processing is executed using MATLAB.

Listing 4.1 shows the target code in Lines 12-17, in which all the bits in Accumulator A are toggled from 0x00 to 0xFF and back several times. This code is expected to have a strong power signature given the relatively large number of bits switching at once.

```
0   clra                    ;a = 00
    back: ldab    #$ff      ;LEDs ON
          stab    portb
          nop
          nop
5         nop
          nop
          nop
          ldab    #$00      ;LEDs OFF
          stab    portb
10        nop
          eora    #$ff      ;a = ff
          ldaa    #$00      ;a = 00
          oraa    #$ff      ;a = ff
          eora    #$ff      ;a = 00
15        ldaa    #$ff      ;a = ff
          clra              ;a = 00$
          nop
          nop
           ;.
20         ;. x 15
           ;.
          nop
          nop
          jmp     back
```

Listing 4.1: Precharacterization Code

[2]This material first appeared in "Power Fingerprinting in SDR and CR integrity assessment" [1], ©IEEE, 2009, and is used here with permission.

Before the target code, in lines 2-10, there is a "marker" where the board's LEDs are turned on and off. This creates a current drain offset used to trigger the oscilloscope and provides timing information for feature extraction. After the target code, there is a sequence of "nops"[3] to create a delay before repeating the loop.

A trace captured from a single run of this code provides the fingerprint used to determine the execution of the target code. The captured power trace is shown in Fig. 4.1, where the current offset due to the marker indicates the beginning of the loop. Using cycle information from the processor's documentation, the segment of the trace corresponding to the execution of the target code is determined (the markers at the bottom of the plot indicate the estimated processor clock cycles). After filtering out the low frequency components with a high-pass filter with cutoff frequency of 1.5 MHz, the specific segment used as the power fingerprint is extracted.

By cross-correlating this power signature

$$S = \{s_0, s_1, \ldots, s_{L-1}\}$$

against traces captured from different runs of the same code

$$R = \{r_0, r_1, \ldots, r_{K-1}\};\ K \geq L$$

the signature is more easily identified. The cross correlation for different sample lags, k, of the traces is given by:

$$\rho_{SR}(k) = \frac{\sum_{n=0}^{L-1} s_n r_{k+n} - L\bar{S}\bar{R}}{(L-1)\sigma_S\sigma_R}; \qquad 0 \leq k \leq K - L$$

where $\bar{S}$ and σ_S are the sample mean and standard deviation of S, respectively, and $\bar{R}$ and σ_R are the sample mean and standard deviation of the subsequence $\{r_{k+0}, r_{k+1}, \ldots, r_{k+L-1}\} \in R$.

Figure 4.2 shows the correlation of the original power fingerprint with the same trace from where it was extracted and with subsequent runs of the same code. The correlation spikes

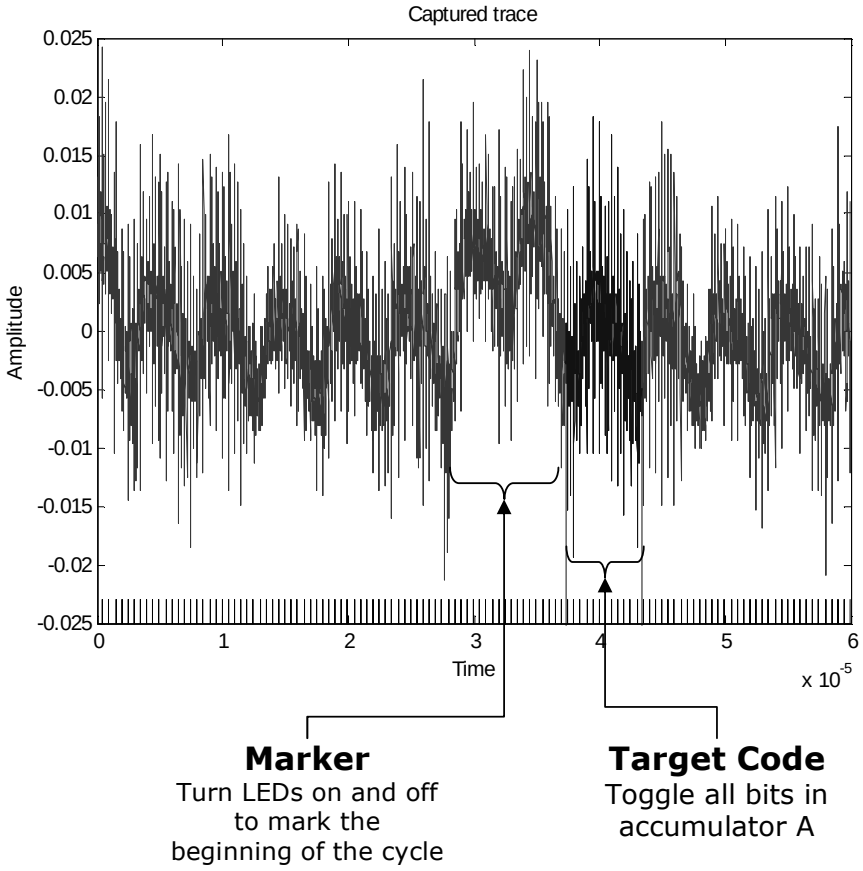

Figure 4.1: Captured Precharacterization Power Trace

appearing at the expected loop repetition rate indicate the execution of the target code.

A different experiment in [1] also demonstrates the ability of PFP to determine when the target code is not executed. To test this, a modified version of the target code is developed with the same functionality (i.e. the contents of Accumulator A are toggled in the same order and the same number of times) but using a different set of instructions, as shown in Listing 4.2. These modifications maintain most of the intrinsic periodicities of the code (the frequency of the loop and the duration of the target code) as well as logical behavior, increasing the risk of incorrect identification by our approach.

[3]The NOP instruction means "no operation" and no registers, other than the program counter (PC), are affected by it.

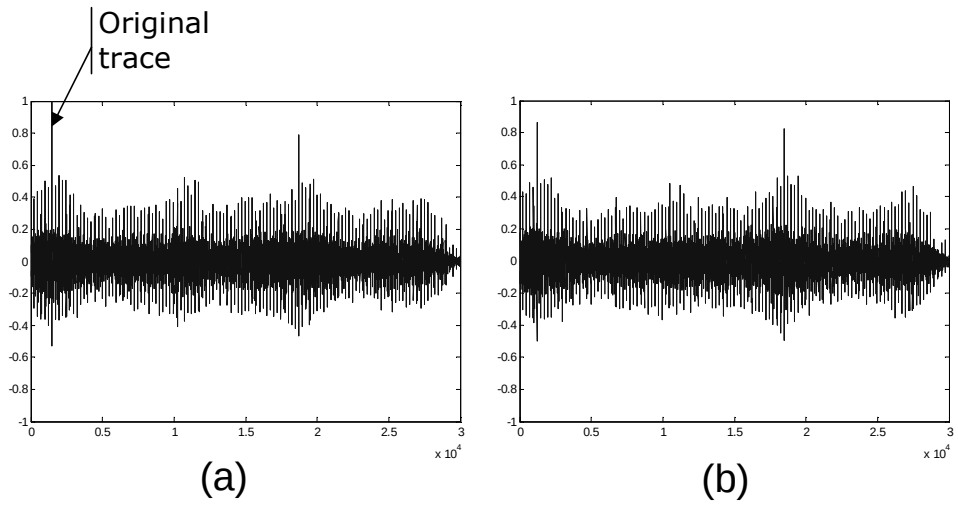

Figure 4.2: Cross-correlation results with traces from the original (a) and subsequent runs (b)

When cross-correlating the original signature with traces captured from the execution of the modified code we notice a lack of strong peaks, as shown in Fig. 4.3(a). This indicates that the original code did not execute. When correlating with a power signature obtained from the new set of instructions, however, the strong peaks show up again, as seen in Fig. 4.3(b). This corroborates the correct identification of target code execution.

```
    oraa    #$ff    ;a = ff
    clra            ;a = 00
    ldaa    #$ff    ;a = ff
15  anda    #$00    ;a = 00
    eora    #$ff    ;a = ff
    ldaa    #$00    ;a = 00$
```

Listing 4.2: Modified Loop Instructions

4.3 Feasibility on Commercial Deterministic Platforms

One of the remaining concerns from the early PFP results, summarized in the previous section, is the feasibility of the approach on basic commercial platforms running practical applications. As mentioned before, those results use a controlled environment and a routine

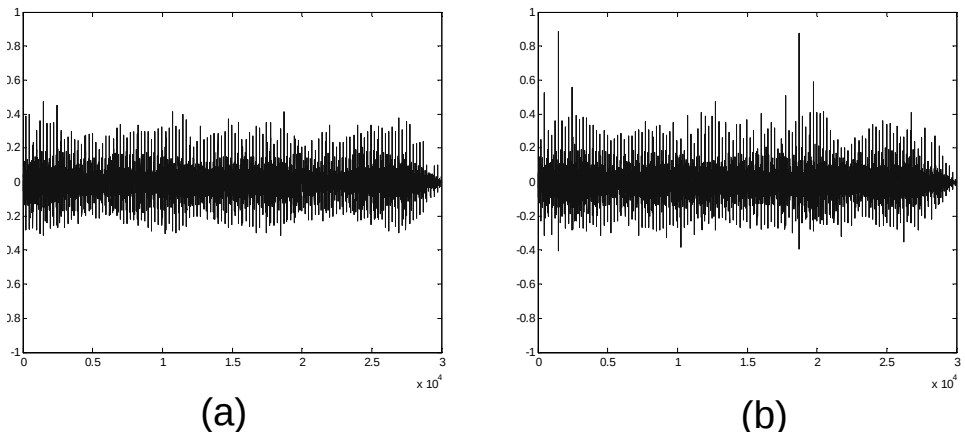

Figure 4.3: Cross-correlation results with different instructions in loop (a) and with a signature obtained from the modified code (b)

designed to yield strong power signatures and not to accomplish a specific task. This section provides results from feasibility experiments that demonstrate the ability of PFP to monitor the execution of production-grade software routines in a representative commercial platform used in communication systems.

4.3.1 Platform Description

The representative target platform is a commercial PICDEM Z Demonstration Kit from Microchip Technology Inc. [46]. This kit is intended as an evaluation and development platform for IEEE 802.15.4 [36] application designers. This standard describes the physical (PHY) and media access control (MAC) layers for low-rate wireless personal area networks (WPANs). In the 2450 MHz band, the standard defines a PHY layer using direct-sequence spread spectrum (DSSS), supporting an over-the-air data rate up to 250 kb/s. The 802.15.4 standard is the basis for the ZigBee and MiWi specifications, which define upper layers not covered in the standard to provide a complete networking solution. The kit includes a motherboard with a PIC18LF4620 8-bit microcontroller, a MRF24J40MA daughterboard with RF transceiver and antenna, and three different software stacks for Zigbee, MiWi, and

MiWi P2P.

The MRF24J40MA in its **Normal Operational Mode** implements the IEEE 802.15.4 specifications PHY layer. It employs a 16-ary quasi-orthogonal modulation technique and direct-sequence spread spectrum with a chip rate of 2000 Mchips/s. Four information bits are used to select one of 16 nearly-orthogonal pseudo-noise sequences (PN) to be transmitted. The sequences are 32 bits long yielding a spreading gain of 8. The PN sequences for successive data symbols are concatenated, and the aggregate chip sequence is modulated onto the carrier using offset quadrature phase-shift keying (O-QPSK). There is another operational mode for the MRF24J40MA which allows higher throughput, but it is not compliant with the standard, known as the **Turbo Operational Mode**. More details on this mode are presented later in this chapter as they are useful to understand our feasibility experiments.

The MiWi P2P stack provides the IEEE 802.15.4 seven security modes [78] and the appropriate API's to use the encryption module of the MRF24J40MA. The security modes can be categorized into three groups:

- AES-CTR mode: Encrypts the message with a 16-byte security key. The mode has no built-in message integrity or frame freshness check
- AES-CBC-MAC modes: Ensures the integrity of the message with 4/8/16 bytes of Message Integrity Code (MIC) attached to each packet. The payload, however, is not encrypted.
- AES-CCM modes: Combines the previous two security modes

4.3.2 PIC18 Instruction Cycle and Pipelining [47]

The PIC18 included in the PICDEM Z platform can be clocked by an external source or an internal oscillator. Independent of the source, however, the clock input is internally divided by four to generate four non-overlapping quadrature clocks (Q1, Q2, Q3 and Q4). The

Program Counter (PC) is incremented on every Q1 while the instruction is fetched from the program memory and latched into the instruction register during Q4. The instruction flows through the pipeline and is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow are shown in Figure 4.4 [47].

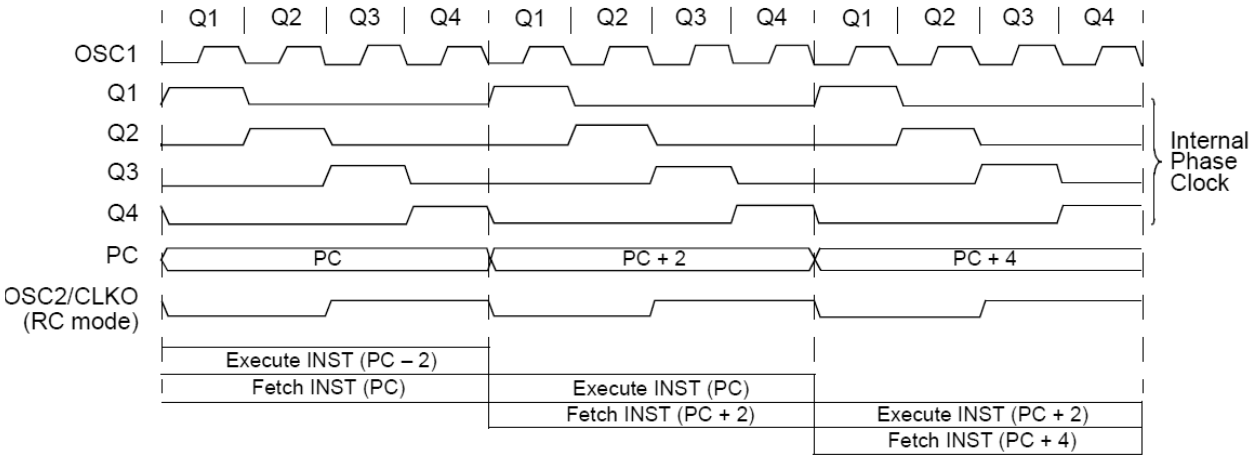

Figure 4.4: PIC18 instruction cycle timing [47].Used under fair use guidelines, 2011.

In the PIC18, an "Instruction Cycle (T_{CY})" consists of four Q cycles: Q1 through Q4. As mentioned before, the instruction fetch and execute are pipelined in such a manner that a fetch takes one instruction cycle, while the decode and execute take another instruction cycle. Therefore, thanks to pipelining, each instruction effectively executes in one intruction cycle, as shown in Figure 4.5. If an instruction causes the PC to change (e.g., BRANCH), then two cycles are required to complete the instruction. All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed. A fetch cycle begins with the PC incrementing in Q1. In the execution cycle, the fetched instruction is latched into the Instruction Register (IR) in cycle Q1. This instruction is then decoded and executed during the Q2, Q3 and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).

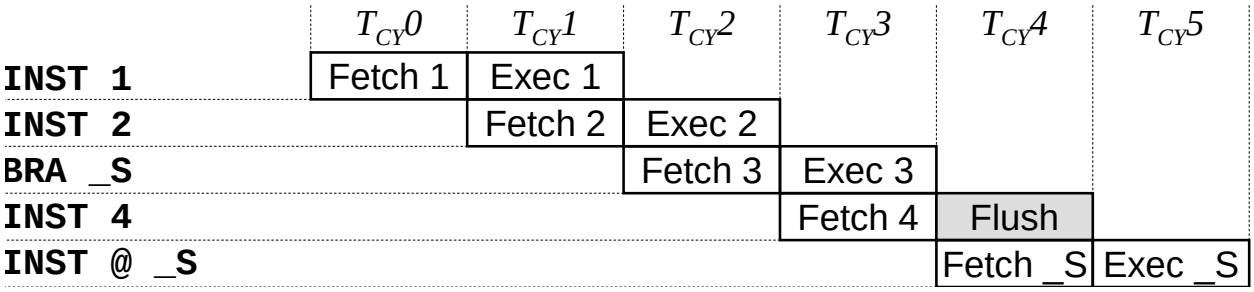

Figure 4.5: PIC18 instruction pipelining

4.3.3 Measurement Setup[4]

Trace collection is performed using a Tektronix TDS 649C real-time oscilloscope and a Tektronix CT-6 current probe. The probe is connected right after the voltage regulators on the motherboard, which includes provisions for taking these measurements. The oscilloscope is configured to 500 MS/s and 10 mVΩ. The trigger is configured to external source (driven by an LED on the board), falling-edge, 40 mV level, and no pre-trigger samples are kept. A total of $L = 30,000$ samples are collected after every trigger event. The experiment setup is depicted in Fig 4.6. Two hundred traces are captured from the execution of each scenario and transferred to a host computer using GPIB for their posterior analysis in MATLAB.

The board itself was slightly modified to let power consumption features be captured. From the motherboard, a total of six decoupling capacitors were removed totaling a cumulative 6 μF. The function of these capacitors is to mitigate the stress placed on the power supplies by the strong current peaks caused by digital processors. It is important to note that removing decoupling capacitors would not be necessary if the provisions to connect the sensor are placed closer to the processor power pins.

A sample trace capture using this setup is shown in Figure 4.7. In this plot, the separate instruction cycles (T_{CY}) are clearly seen as a set of four shorter Q cycles. In the figure a total

[4]The material in this section first appeared, though in a different form, in "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010, and is used here with permission.

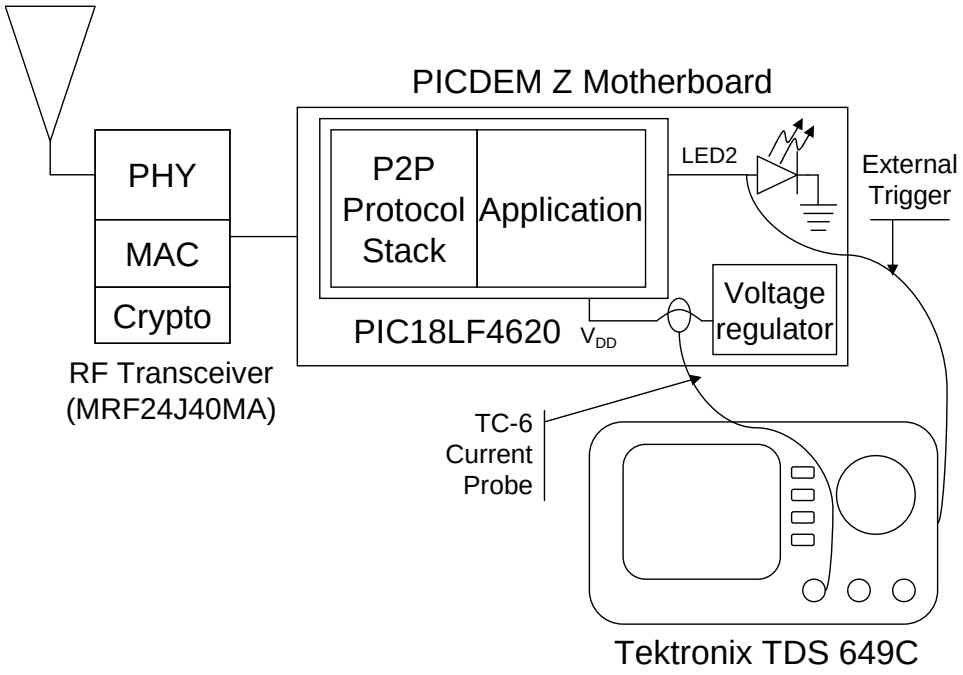

Figure 4.6: Measurement Setup.

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010. Used with permission.

of 10 instruction cycles are displayed. It is very important to notice that, due to an internal PLL that increases the clock frequency by a factor of 4, there are four possible starting points for the instruction cycle that align differently with the external clock input, which results in slightly different traces. This issue can be overcome by starting the processor with the PLL off until the processor has started, and then turning it on. This way, the instruction cycles will always align with the clock source the same way and the traces have a similar look.

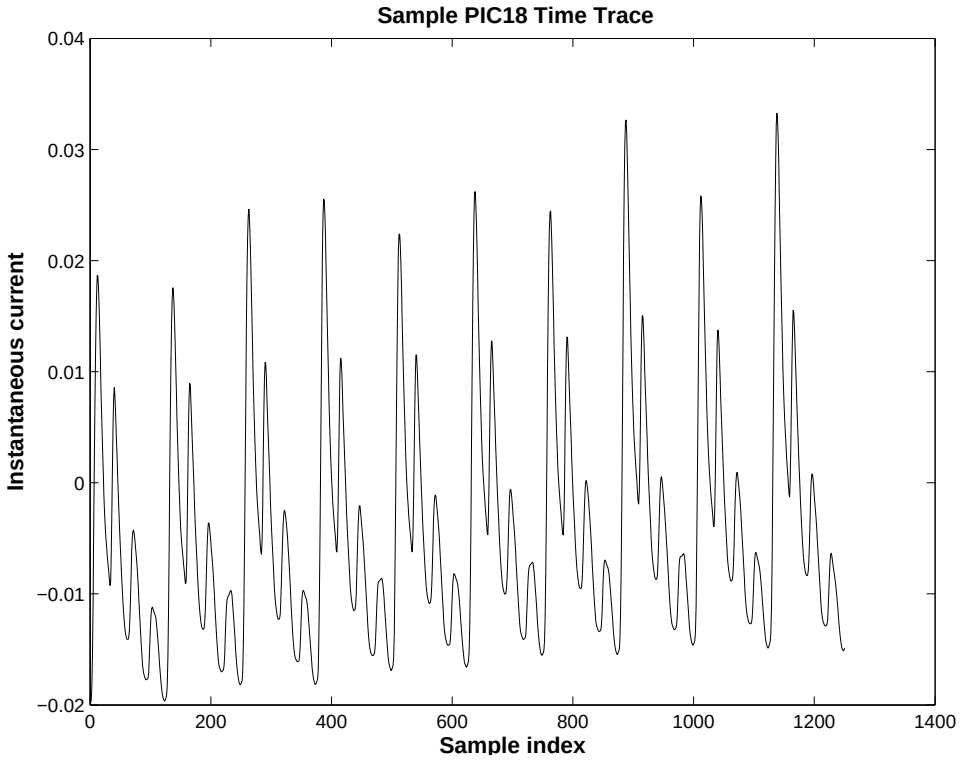

Figure 4.7: PIC18 sample time traces

4.3.4 Trace Analysis[5]

For this work, traces of length L captured during the ith execution of code α are represented by (4.1)

$$r_{\alpha}^{(i)}[n]\,; \qquad n = 0, \ldots, L-1 \tag{4.1}$$

Because the current probe is placed right after the voltage regulators, other board components introduce interference to the measurements. Particularly, the RF transceiver introduces a low frequency, but strong, oscillation due to charge pumps. Because software-related power consumption is characterized by sharp transitions, the first difference between traces, given by (4.2), is used.

[5]The material in this Section first appeared, though in a different form, in "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010, and is used here with permission.

$$d_{\alpha}^{(i)}[n] = r_{\alpha}^{(i)}[n] - r_{\alpha}^{(i)}[n-1] \tag{4.2}$$

By using the sample difference, sharp transitions are preserved while slow changing elements are greatly reduced. This operation is effectively a multiplication-less high-pass filter. Captured traces from the execution of trusted code α are used to create a signature, our target fingerprint, s_{α}. The signature is given by (4.3), where N traces are averaged to form the target signature and reduce the effects of random noise in our measurements.

$$s_{\alpha}[n] = \frac{1}{N}\sum_{i=0}^{N-1} d_{\alpha}^{(i)}[n]\,; \qquad n = 0, \ldots, L-1 \tag{4.3}$$

For these experiments, the process of extracting discriminatory features from the execution of unknown code β consists of simple time-domain correlation against s_{α}. The correlation, however, is performed on J partial sections of the signature and the trace, each section has a length $w = \lfloor L/J \rfloor$. This partial correlation is performed to avoid spreading potential differences in the power traces across the full trace.

The cross correlation for different sample lags, $0 \leq k \leq w$, of section $j = 1, 2, \ldots, J$ of the traces is given by:

$$\rho_{s_{\alpha}d_{\beta}^{(i)}}(j,k) = \frac{\sum_{n=(j-1)w}^{jw} s_{\alpha}[n]d_{\beta}^{(i)}[k+n] - w\bar{s}\bar{d}}{(w-1)\sigma_s\sigma_d}, \tag{4.4}$$

where $\bar{s}$ and σ_s are the sample mean and standard deviation of the corresponding section in s_{α}, and $\bar{d}$ and σ_d are the sample mean and standard deviation of the corresponding section in $d_{\beta}^{(i)}$.

In order to compensate for any clock drifts, we keep the maximum correlation values for different lags. This action described in (4.5) reduces the dimensionality of our traces to only a sequence of J peak correlation values for every trace.

$$\hat{\rho}_{s_{\alpha}r_{\beta}^{(i)}}(j) = \max_{k}\left\{\rho_{s_{\alpha}r_{\beta}^{(i)}}(j,k)\right\} \tag{4.5}$$

Under ideal noiseless conditions and with $\beta = \alpha$, $\hat{\rho}_{s_\alpha r_\beta^{(i)}}(j) = 1$ for every section j. Any deviation from the power consumption characteristics would be reflected by a reduced correlation factor. The final test statistic or discriminatory feature used in this work is the minimum peak correlation value for that specific trace, which is given by (4.6),which indicates the maximum deviation from the signature of instance i of code β.

$$x_\beta^{(i)} = \min_j \hat{\rho}_{s_\alpha r_\beta^{(i)}}(j) \tag{4.6}$$

Using random variable (4.7), for every captured trace i we can design appropriate detectors using different criteria depending on the statistical information we can gather from the system a priori.

$$X_\beta = x_\beta^{(i)} \tag{4.7}$$

In summary, after capturing traces, we divide them into J subsections, align each section against the corresponding section of the target signature, calculate the correlation coefficient between them, determine the maximum deviation from the signature, and make a decision whether the traces correspond to the execution of the target code.

Detector Design

From the resulting distributions of X_α and X_β we can design a likelihood ratio detector. The goal is to discriminate between two hypothesis:

- H_α: traces match the execution of trusted code α, and
- H_o: traces do not match the execution of trusted code

In a likelihood ratio, the decision rule based on a-posteriori probabilities is to choose H_o when

$$\frac{p_o(X)}{p_\alpha(X)} \geq \lambda \tag{4.8}$$

Where $p_\alpha(X)$ is the true probability distribution of X_α and $p_o(X)$ is the probability distribution of the minimum peak correlation values of the alternative code, in this case X_β. An optimal λ can be determined given complete a priori information about $\Pr(H_\alpha)$ and $\Pr(H_o)$, the true probabilities that our target code is executing or not. In our case, however, we do not have such information and instead we apply the Neyman-Pearson criteria to design a detector.

If we represent choosing H_α or H_o with D_α and D_o, respectively, then the optimal λ would maximize $\Pr(D_\alpha|H_\alpha)$ while minimizing the probability of making an error. The Neyman-Pearson criterion allow us to determine a decision threshold that maximizes the probability of correct detection while maintaining a given probability of false alarm, $\Pr\{D_o|H_\alpha\}$ [75].

Notice that (4.8) is equivalent to choosing H_o if $X \leq \gamma$, where γ is chosen to meet a given probability of false alarm. $\Pr\{D_o|H_\alpha\}$ is equal to the left tail probability in $p_\alpha(X)$ from threshold γ.

4.3.5 Feasibility Experiment: Regulatory Application[6]

In our first feasibility experiment we aim to determine the feasibility of using PFP to detect unauthorized software modifications that can affect the spectral emissions of SDR. As mentioned before the PICDEM Z platform we use in this work can operate in two modes: Normal and Turbo. The operational mode is set by programming a set of registers in the transceiver. In the Normal Mode, the transceiver implements the IEEE 802.15.4 specifications PHY layer supporting an over-the-air data rate up to 250 kb/s. Figure 4.8 shows a

[6]The material in this Section first appeared, though in a different form, in "POWER FINGERPRINTING IN UNAUTHORIZED SOFTWARE EXECUTION DETECTION FOR SDR REGULATORY COMPLIANCE" [3], ©WInnForum, 2010, and is used here with permission.

snapshot of the spectrum usage during a Normal Mode transmission. Normal Mode is the default configuration and the configuration routines that set this mode are considered our reference or trusted code.

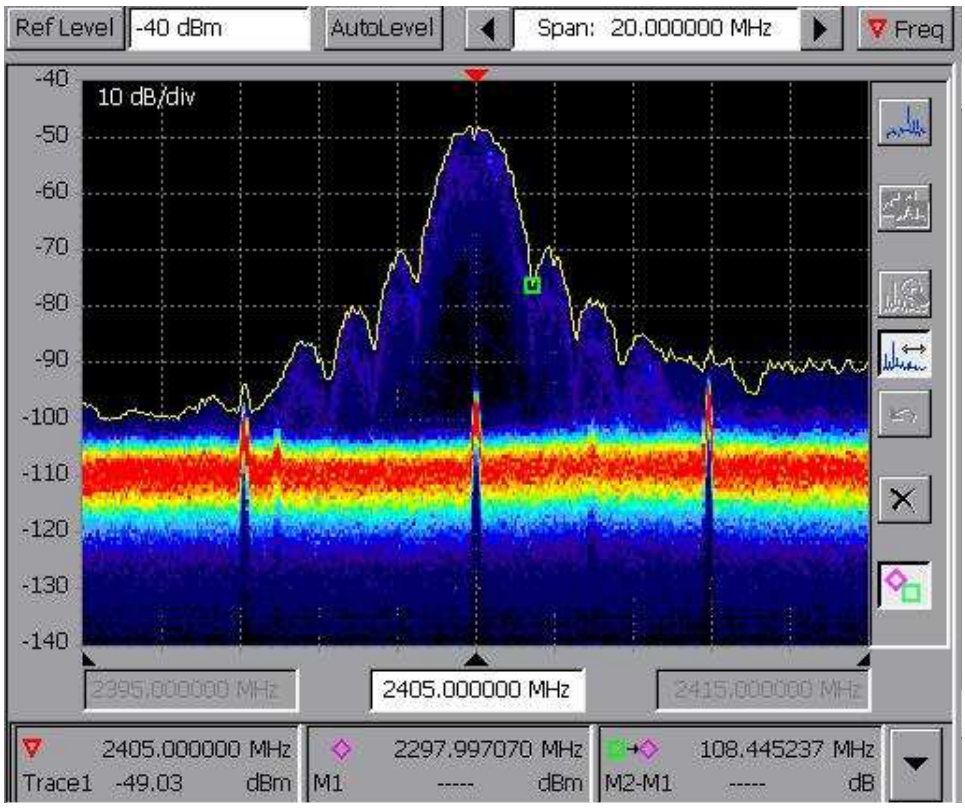

Figure 4.8: Normal mode spectral occupancy

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "POWER FINGERPRINTING IN UNAUTHORIZED SOFTWARE EXECUTION DETECTION FOR SDR REGULATORY COMPLIANCE" [3], ©WInnForum, 2010. Used with permission.

Turbo Mode can deliver a maximum throughput of 625 kbps. This is an increase of 2.5 times the Normal Mode. The Turbo Mode is not compliant with the IEEE specifications and no information is given about the specific modifications that implement Turbo Mode. After observing the occupied spectrum, however, it is clear the throughput improvements come as a result of wider spectrum usage, as can be seen in Figure 4.9, where the spectrum

occupancy is also roughly 2.5 time the occupancy in the Normal Mode.

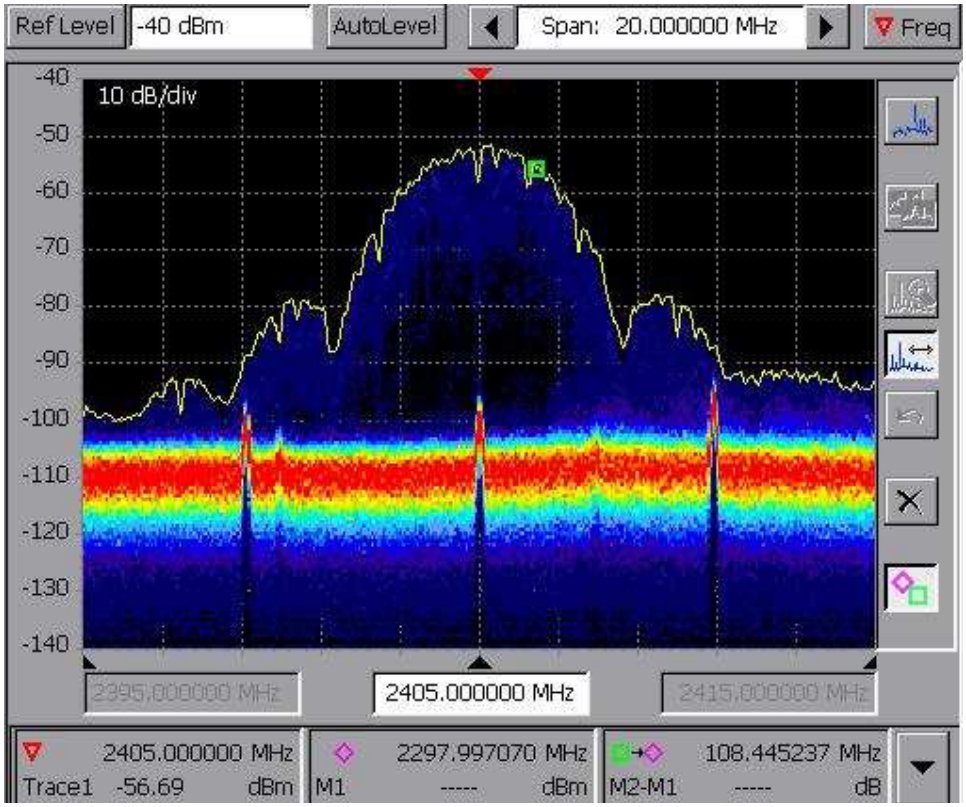

Figure 4.9: Turbo mode spectral occupancy

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "POWER FINGERPRINTING IN UNAUTHORIZED SOFTWARE EXECUTION DETECTION FOR SDR REGULATORY COMPLIANCE" [3], ©WInnForum, 2010. Used with permission.

Both configurations of the MRF24J40MA RF module are used in our experiment to test whether PFP can detect execution changes that impact spectral emissions. We consider the following scenario: A product using the MRF24J40MA is approved for market release by a regulatory body under the Normal Mode and a power fingerprint is obtained at certification time. A savvy user, or an insider, (the "attacker") looks to benefit from higher throughput and reconfigures the system to use Turbo Mode. This modification affects spectral emissions, invalidates the regulatory certification granted, and can cause interference to other users.

The actual attack model is irrelevant in PFP. So, we assume that the attacker has access to the source code and a way to reprogram the processor controlling the RF module. A PFP monitor capturing run-time traces is expected to determine whether the execution corresponds to the code used during certification or not.

Operational Mode Profiling Software

The software structure shown in Listing 4.3 is used to configure both operational modes. In this routine, the board is initialized, a connection is established, the RF transceiver sends a packet using the UnicastConnection API, and a software reset is performed after a short delay to start the routine all over again.

```
BoardInit();
ConsoleInit();
P2PInit();
SetChannel(myChannel);
EnableNewConnection();
CreateNewConnection();
FlushTx();
//write payload data
UnicastConnection(0, //ID
                  FALSE, //Cmd?
                  TRUE);//crypto?
//delay
_asm
    RESET  //Software Reset
_endasm
```

Listing 4.3: MiWi P2P Profiling Code

Within the P2PInit() implementation, there is a call to another function called MRF24J40Init() at the end of which the code shown in Listing 4.4 is included. The three lines writing to the different baseband registers (BBREG0, BBREG3, BBREG4) in the MRF24J40 configure it in the Turbo Mode. The last two lines perform a reset of the RF state machine so the changes are accepted by the MRF24J40. The default configuration of the MRF24J40 upon reset is the Normal Mode.

```
    .
    .
    .
TMR0H = 0x00; //Clear Timer0 high byte
TMR0L = 0x00; //Clear Timer0 high byte
LED_2 = 1;    //Falling edge in LED2
LED_2 = 0;    //to trigger oscilloscope

#ifdef TURBO_MODE
    PHYSetShortRAMAddr(WRITE_BBREG0, 0x01);
    PHYSetShortRAMAddr(WRITE_BBREG3, 0x38);
    PHYSetShortRAMAddr(WRITE_BBREG4, 0x5C);

    PHYSetShortRAMAddr(WRITE_RFCTL,0x04);
    PHYSetShortRAMAddr(WRITE_RFCTL,0x00);
#endif
```

Listing 4.4: MiWi P2P Configuration Code

Right before configuring the baseband registers we restart Timer 0 in the processor and turn on and off LED2 on the board. Timer 0, a 16-bit timer, is used by the MiWi P2P software stack and is always running. By reinitializing it before performing a transmission we have a more deterministic power signature without affecting the operation of the system. LED2 is used to trigger the oscilloscope to start collecting traces at the beginning of the transmission code execution.

We test two different scenarios. In Scenario 1, we aim to discriminate between code compiled with and without the `TURBO_MODE` definition. The attacker just has to recompile using the `TURBO_MODE` directive to set up the Turbo Mode. The authorized version does not include the code within the `#ifdef` section.

Notice that in Scenario 1, extra instructions are inserted for configuring the Turbo Mode. This makes the job of the PFP monitor easy, as the insertion changes instructions, fetches, and parameters, therefore yielding a much different fingerprint.

In Scenario 2, however, the Normal Mode configuration is explicit. The `#ifdef` directives are removed and the Normal Mode is configured by setting the values of BBREG0, BBREG3, BBREG4 to 0x00, 0xD8, and 0x9C, respectively. The attacker still has to write the baseband registers with the values in Listing 4.4 to set the Turbo Mode. Scenario 2 makes PFP more

challenging, as there are no instructions added, just a change in the parameters.

In order to make power trace capture of the configuration code more efficient, we execute a software reset, after a short delay, once the unicast transmission is completed. This does not affect the execution of the profiled code. **It is very important to keep in mind that monitoring takes place during the execution of the configuration sequence only, not during the actual radio transmission.**

Operational Mode Configuration Results

An example of a single differential trace, $d_{\alpha}^{(i)}$, and the averaged signature are shown in Figure 4.10. On the detail window shown in the same figure, we can easily see pronounced spikes characteristics of digital processors dynamic power consumption.

For Scenario 1, the average peak correlation values from $N = 100$ execution instances, given by (4.9), are shown in Figure 4.11.

$$\bar{\rho}_{s_\alpha r_\beta}(j) = \frac{1}{N}\sum_{i=1}^{N} \hat{\rho}_{s_\alpha r_\beta^{(i)}}(j) \tag{4.9}$$

There is a noticeable difference between the correlation with traces from the same configuration code, Normal Mode (α), and those from different configuration code, Turbo Mode (β). As expected, the average peak correlation value is higher for traces from the configuration of Normal Mode. Additive noise prevents a perfect correlation in this case.

The distribution of the minimum peak correlation values for traces captured from configuration in both modes Normal and Turbo, X_α and X_β, is given in Figure 4.12. The separation between sample distributions clearly indicates the ability of PFP to discriminate between them. Notice that not a single sample overlaps to the opposite distribution.

Having these sample distributions, it is possible to design a suitable detector using the Neyman-Pearson criterion. The difference between distributions, however, is so large that the probability of making a classification error is negligible. The robust performance of

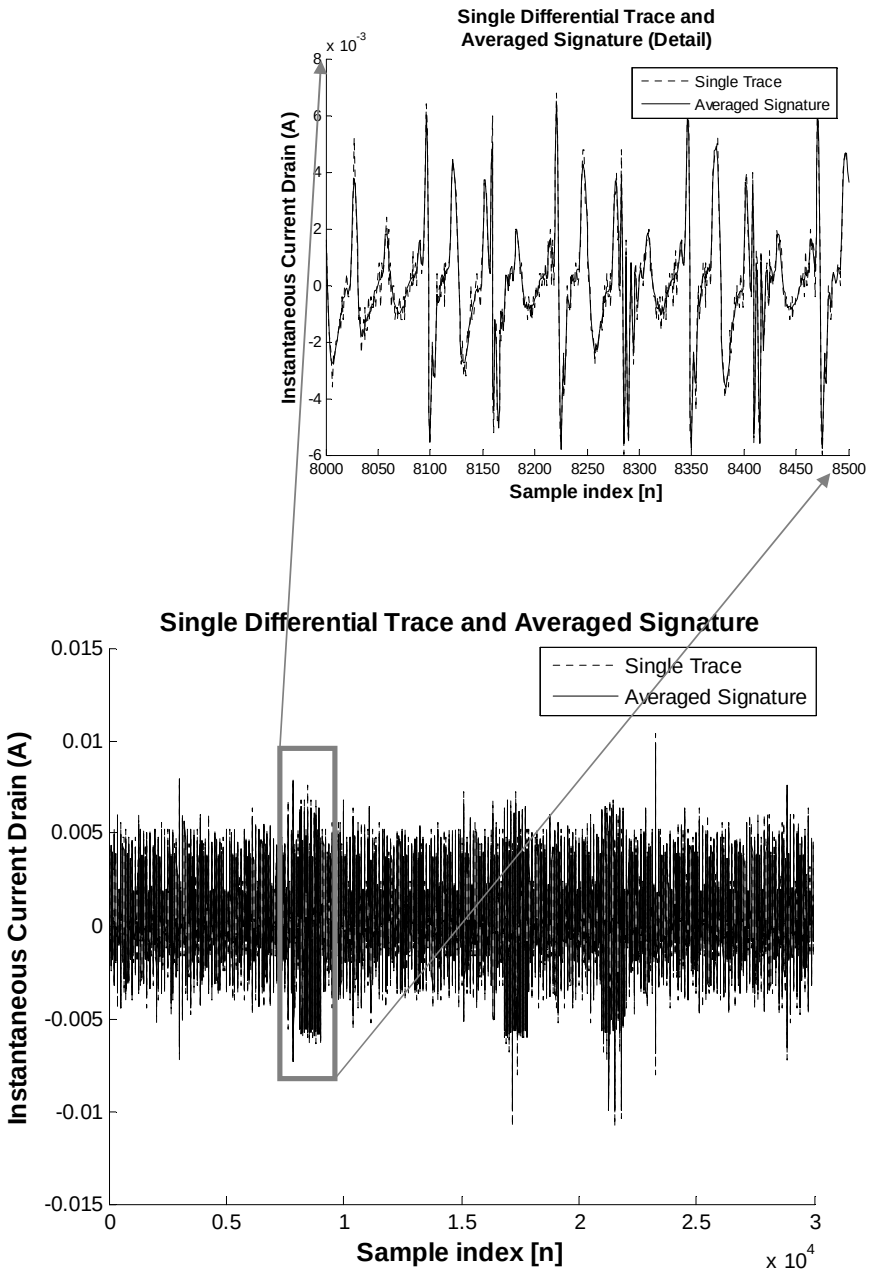

Figure 4.10: Differential Trace and Averaged Signature.

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010. Used with Permission.

PFP in discriminating between both configurations is expected, given that dynamic power consumption depends not only on instructions and inter-instructions transitions but also on

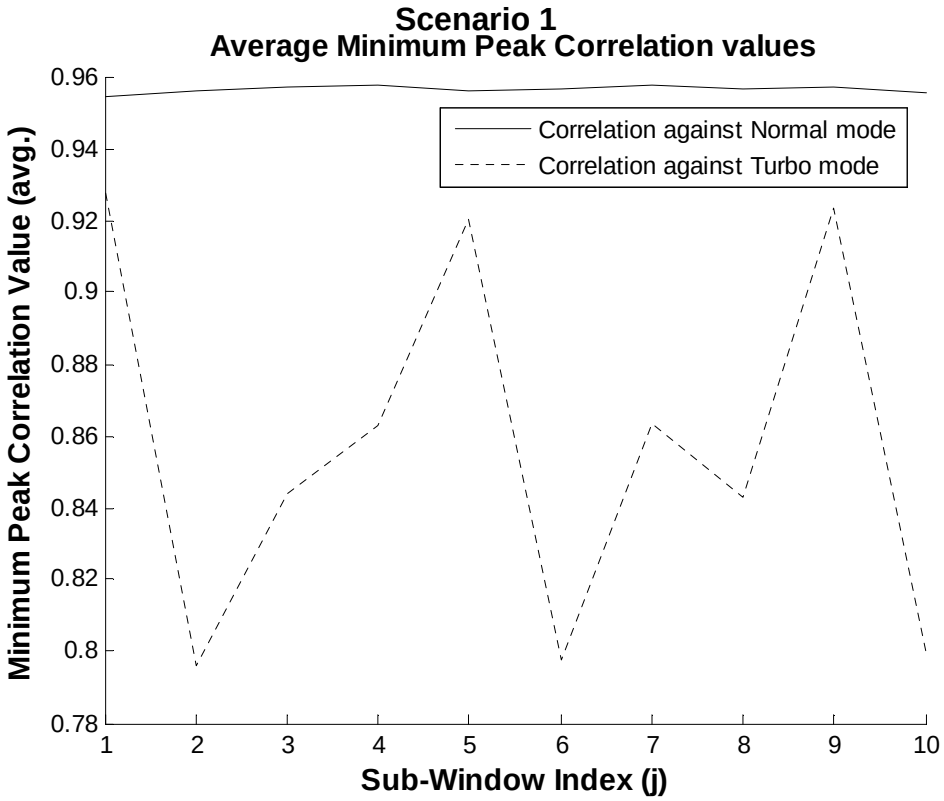

Figure 4.11: Average minimum peak correlation value when original code uses default configuration

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "POWER FINGERPRINTING IN UNAUTHORIZED SOFTWARE EXECUTION DETECTION FOR SDR REGULATORY COMPLIANCE" [3], ©WInnForum, 2010. Used with permission.

addresses and parameters. In Scenario 1, we are causing a large disruption on the execution patterns by adding extra instructions, which requires fetching from different memory locations, different parameters, and different inter instruction transitions. Hence, a noticeable difference is expected.

Scenario 2 is more challenging from the PFP perspective. In this scenario, the Normal Mode configuration is made explicitly. Hence, the calls to write the baseband registers in the RF module are made regardless of whether the Normal or Turbo modes are configured. This

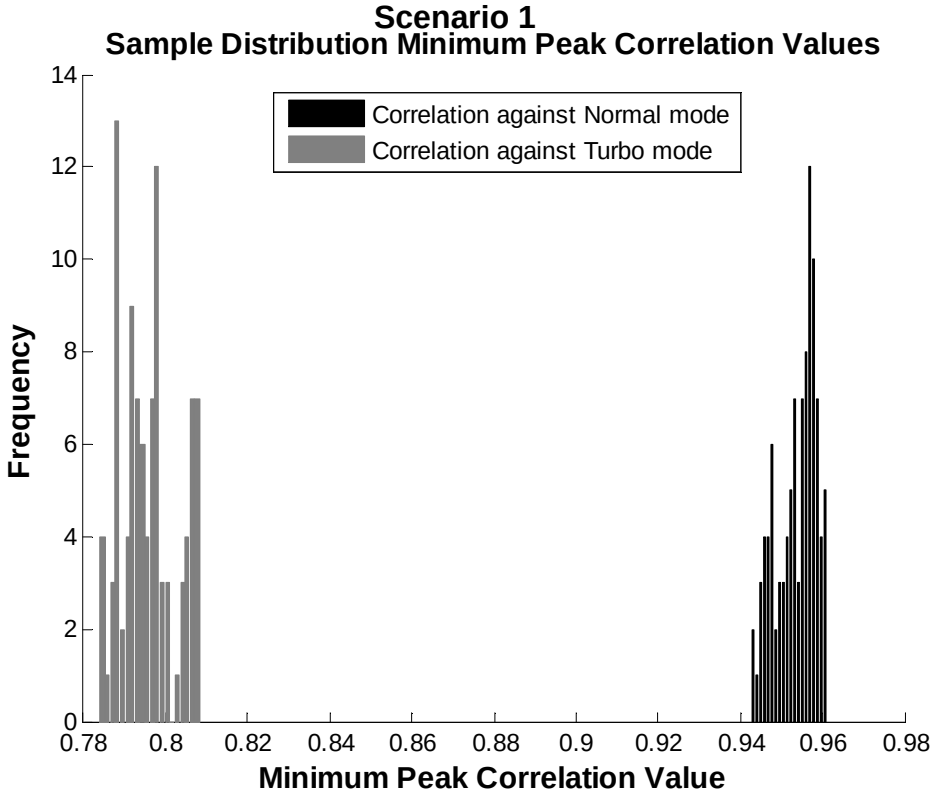

Figure 4.12: Minimum peak correlation values sample distribution when original code uses default configuration

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "POWER FINGERPRINTING IN UNAUTHORIZED SOFTWARE EXECUTION DETECTION FOR SDR REGULATORY COMPLIANCE" [3], ©WInnForum, 2010. Used with permission.

time, there are no instruction or address changes, only parameters are modified. The average minimum peak correlation values of the traces captured in our second scenario are shown in Figure 4.13 and their sample distributions in Figure 4.14

We can see how the distributions are closer than in the previous scenario, but still not a single trace is misclassified. The probability of a classification error using a Neyman-Pearson criterion is once again negligible.

These two scenarios cover a large number of potential attacks. If the attacker manages to

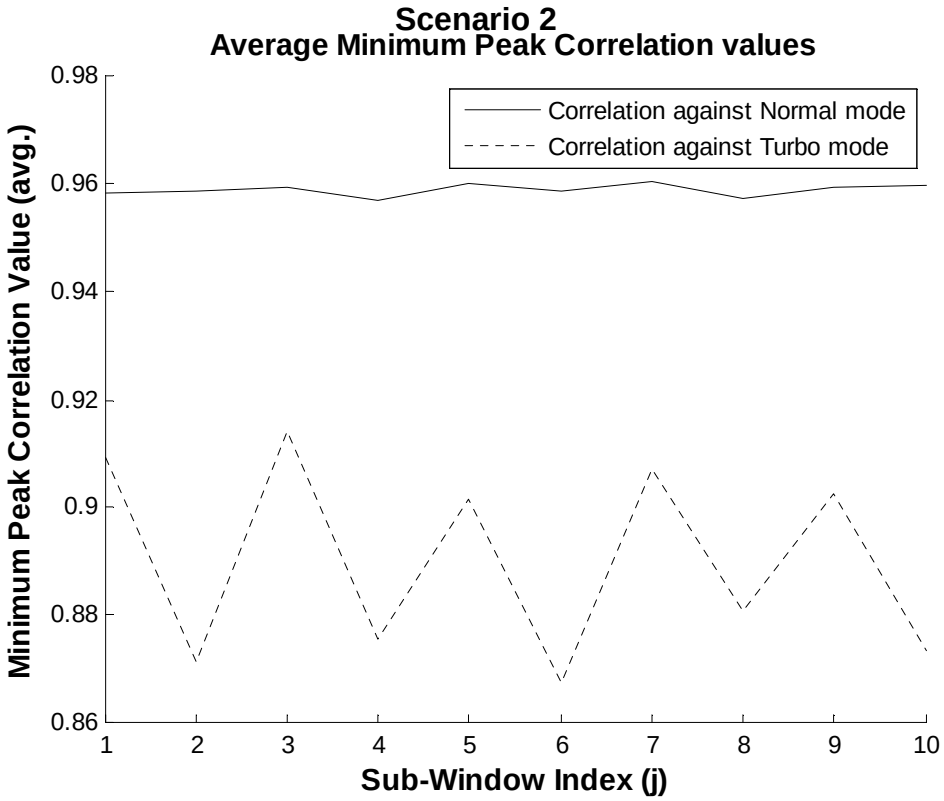

Figure 4.13: Average minimum peak correlation value when original code uses explicit configuration

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "POWER FINGERPRINTING IN UNAUTHORIZED SOFTWARE EXECUTION DETECTION FOR SDR REGULATORY COMPLIANCE" [3], ©WInnForum, 2010. Used with permission.

insert malware by means other than having the source code, it would be a scenario similar to our first experiment. Notice, however, that in order to detect execution deviations, it is necessary to pre-characterize the authorized code, including every execution path. This would be necessary in order to catch an attacker that manages to insert configuration code outside the configuration section.

On the other hand, if an attacker does not require inserting malicious code to configure the RF module to Turbo Mode, as in the case of the explicit configuration used in the second

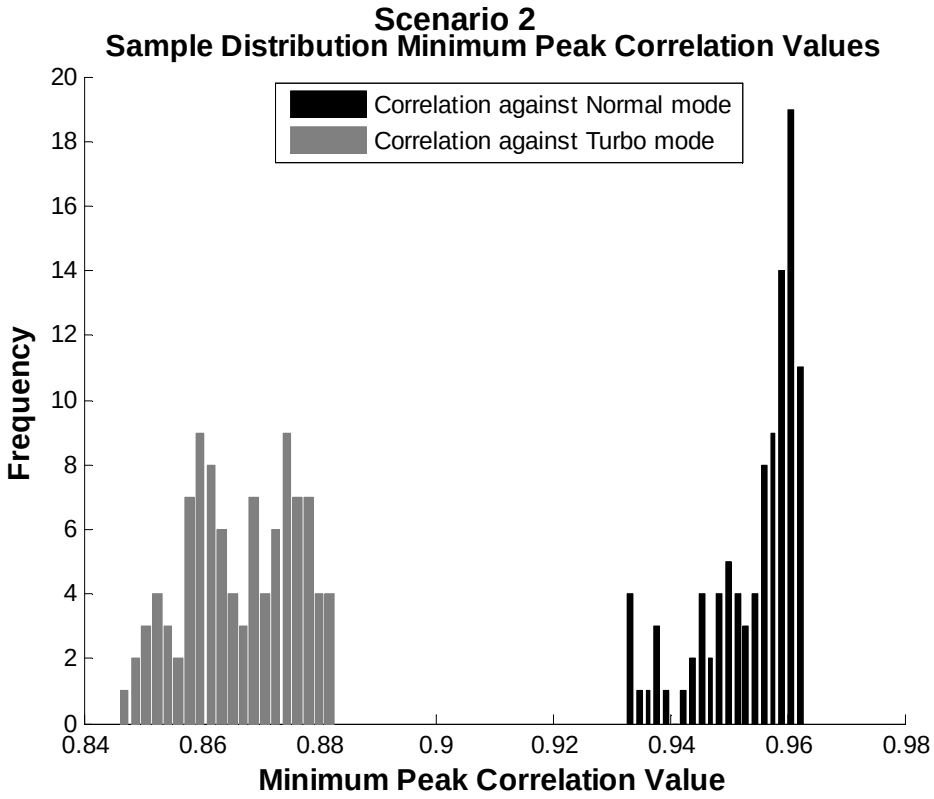

Figure 4.14: Minimum peak correlation values sample distribution when original code uses explicit configuration

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "POWER FINGERPRINTING IN UNAUTHORIZED SOFTWARE EXECUTION DETECTION FOR SDR REGULATORY COMPLIANCE" [3], ©WInnForum, 2010. Used with permission.

scenario, the approach can still detect the execution violations as demonstrated in our second experiment.

4.3.6 Feasibility Experiment: Security and Encryption[7]

The PICDEM Z platform supports seven security settings and provides software APIs to facilitate the usage of the cryptographic modules. As part of the provided API's there are interfaces to transmit data to a single node on a network or to broadcast packets. Included in the arguments for these interfaces, there are options to select if Tx data is to be encrypted by the transceiver. The goal of this experiment is to determine the feasibility of PFP to identify when a security module is bypassed. In this experiment, the interface for unicast transmission is invoked requiring the data to be encrypted, and a signature is extracted during execution. The routine is then modified in two different ways: by invoking a different interface (broadcast) without encryption, and by invoking the same interface with the same data but without requiring the encryption. PFP is used to determine whether test power traces correspond to the execution of the original interface or tampering has occurred. We use the same signal processing approach for trace analysis as the previous experiment and extract the signatures from the trusted execution in the same way.

Security Experiment Profiling Software

We use a simple software structure, similar to the one used in the previous section to test the performance of PFP in different scenarios. The basic functionality of the test routine is to send packets between two nodes continuously with a short delay between transmissions. After initializing the board and establishing a connection with a remote node, we transmit an encrypted packet using the `UnicastConnection` interface and then wait for about one second before sending another packet. Similar to the previous experiment, we also included code to restart Timer 0 and create a trigger using an LED. The code using for this experiment is shown in Listing 4.5.

```
0 BoardInit();
  ConsoleInit();
```

[7]The material in this Section first appeared, though in a different form, in "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010, and is used here with permission.

```
P2PInit();
SetChannel(myChannel);
EnableNewConnection();
CreateNewConnection();
while(1){
  FlushTx();
  //write payload data

  TMR0H = 0x00;    //Clear Timer0 high byte
  TMR0L = 0x00;    //Clear Timer0 high byte
  LED_2 = 1;       //Create a falling edge in LED2
  LED_2 = 0;       //to trigger oscilloscope

  UnicastConnection(0, //ID
                    FALSE,//command
                    TRUE);//crypto
  //delay
}
```

Listing 4.5: Profiling code for security feasibility experiment

The encrypted unicast transmission in Listing 4.5 represents our trusted code and an average signature is extracted from power traces during its execution. Similar to the previous experiment we modify the original routine to create two different scenarios. In Scenario A, we replace the call to `UnicastConnection()` with a call to

```
BroadcastPacket(myPANID, FALSE, FALSE);
```

Notice that with the third parameter we instruct the stack not to encrypt the payload. In this scenario, invoking a different interface creates a large deviation from the execution of the original, trusted routine. This is because a different function is called with the corresponding difference in instructions and addresses. Hence, we expect the difference between traces to be significant

For Scenario B, we create a more challenging environment for PFP. In this case, we use the same interface `UnicastConnection()` and only change the parameter that determines whether the transceiver is to encrypt the payload or not. We replace the original call to unicast with the following call:

```
UnicastConnection(0, FALSE, FALSE);
```

As the same interface is used in Scenario B, we expect the traces to be more similar to the reference signature from the trusted code. A significant difference is still expected because the parameter change forces a different execution path deeper in the stack.

Security Applications Results

Measurements taken from the execution of encrypted unicast transmissions are averaged and used to create the target signature s_α.

For Scenario A, the average peak correlation values from $N = 100$ execution instances are shown in Fig. 4.15. There is a noticeable difference between the correlation with traces from the same transmission type, encrypted unicast (α), and those from different code, unencrypted broadcast (β). As expected, the average peak correlation value is higher for traces from encrypted unicast transmission. Additive noise prevents a perfect correlation in this case.

The distribution of the minimum peak correlation values for all traces captured in both scenarios, X_α and X_β, is given in Fig. 4.16. Once again, the difference in power consumption features is clearly noticeable.

Assuming we set a target $\Pr\{D_o|H_\alpha\} = 0.0001$, following the Neyman-Pearson criterion described before to design a detector, and assuming p_α is Rayleigh with parameter $\sigma = 0.03565$[8], then $\gamma = 0.8675$ would be the optimal threshold. Any $X \leq \gamma$ will be considered as generated by code other than the target. This threshold practically eliminates the possibility of making the opposite error $\Pr\{D_\alpha|H_o\}$.

The robust performance of PFP in discriminating between both types of transmission in Scenario A is expected, because we are invoking two completely different interfaces which will in turn execute different code from different sections in memory. A marked difference in power consumption ensues.

[8]The Rayleigh distribution parameter σ is calculated using a maximum likelihood estimate from $1 - X_\alpha$

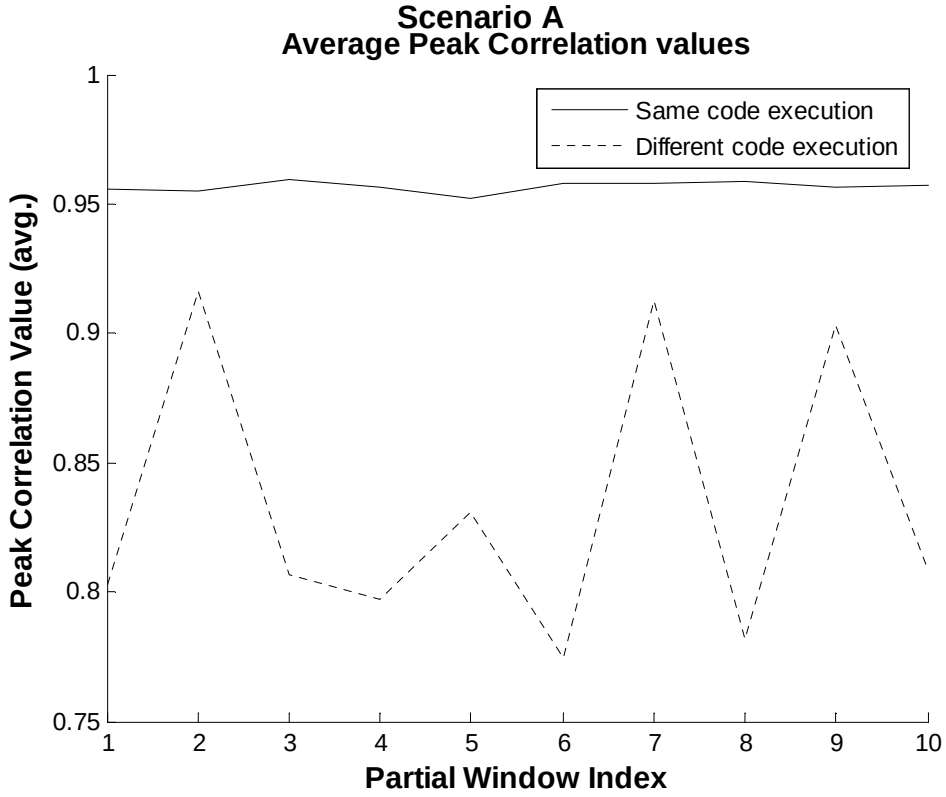

Figure 4.15: Average Minimum Peak Cross-correlation Values Between Encrypted Unicast Tx and Unencrypted Broadcast

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010. Used with Permission.

The more challenging Scenario B is a better representation of what could happen in a real attack, in which a vulnerability could be exploited to change the encryption configuration. Fig. 4.17 shows the average peak correlation values from test traces in this scenario. It can be seen that on average the power consumption of is closer for both types compared to Scenario A, but there is still a clear separation between them.

The sample distributions of the minimum peak correlation values, X_α and X_β, between both scenarios is shown in Fig. 4.18. As expected, the distributions are closer to each other. The same detector can be used but still no single trace in this sample set is miss-classified.

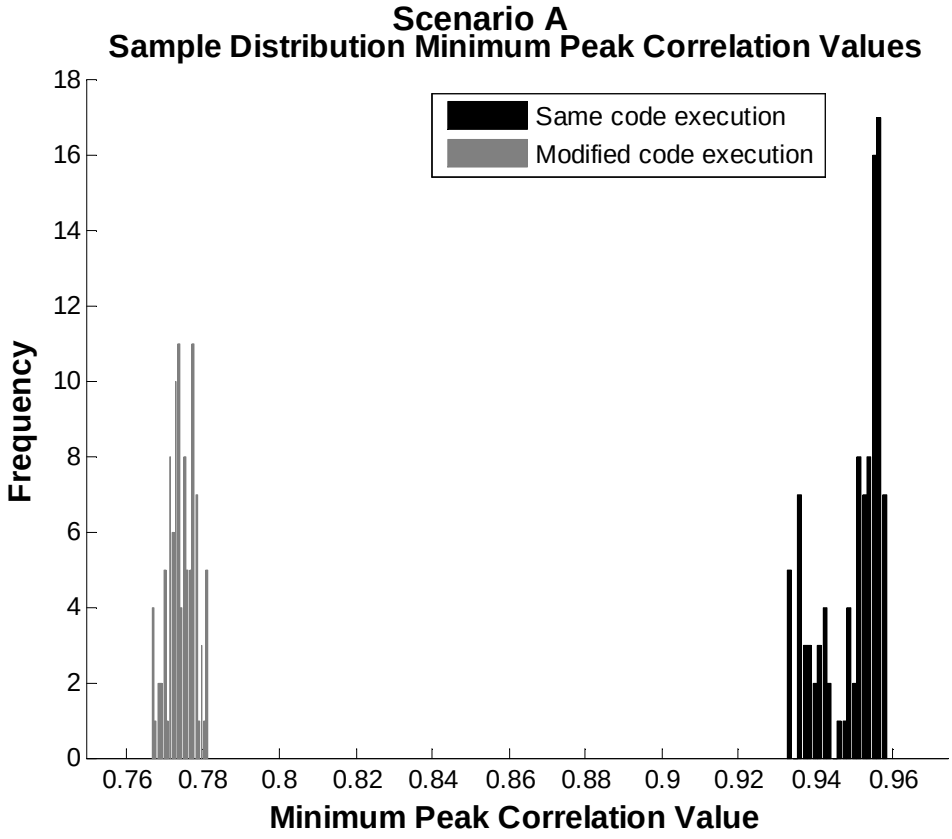

Figure 4.16: Sample Distribution Minimum Peak Cross-correlation Values Between Encrypted Unicast Tx and Unencrypted Broadcast

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010. Used with Permission.

4.4 PFP Sensitivity to Minimum Execution Change[9]

In the previous sections, we have demonstrated the ability of PFP to detect execution deviations resulting from significant violations to expected execution. In this section, we evaluate

[9]The material in this section is presented here with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

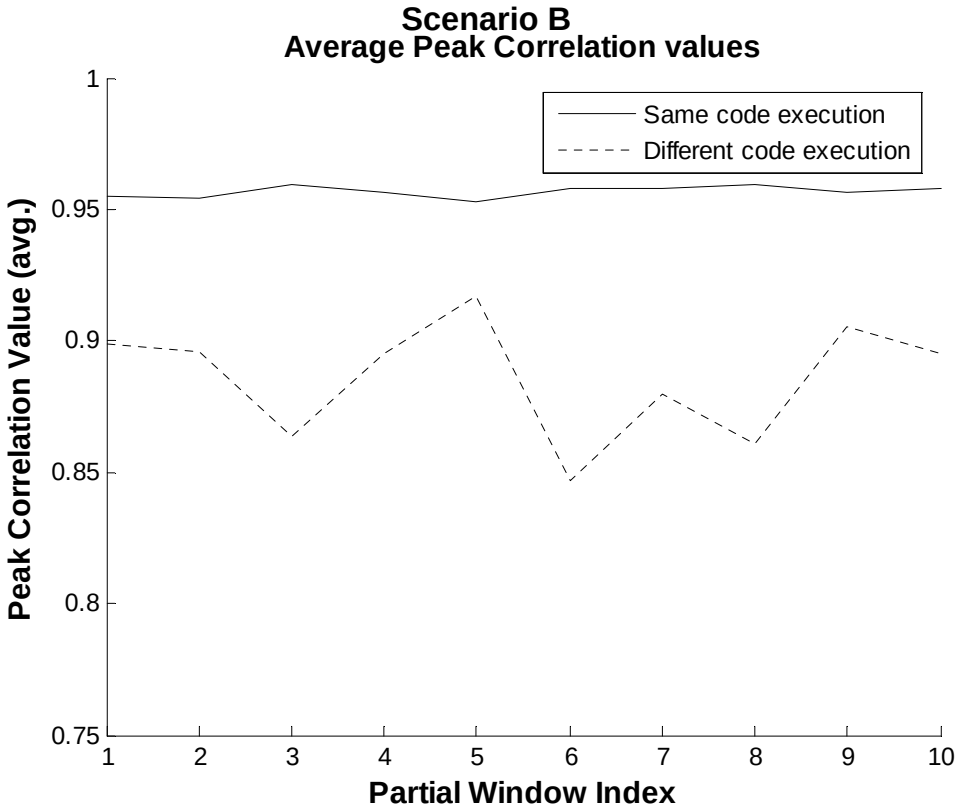

Figure 4.17: Average Peak Cross-correlation Values Between Encrypted and Unencrypted Unicast Tx.

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010. Used with Permission.

the ability of PFP to detect the minimum change possible in execution (from a dynamic power consumption perspective): an execution that is different from the expected signature by a single bit transition. We accomplish this by using time-domain signatures and by characterizing the specific way the processing platform consumes power to concentrate our efforts only on the sections of the traces that carry the most discriminatory information.

The fine-grained measurements used in PFP can lead to redundancies that add little discriminatory information. For example, the traces shown in Fig. 4.19 correspond to a full instruction cycle in the PIC18LF4620 (without the RF daughterboard) when executing two

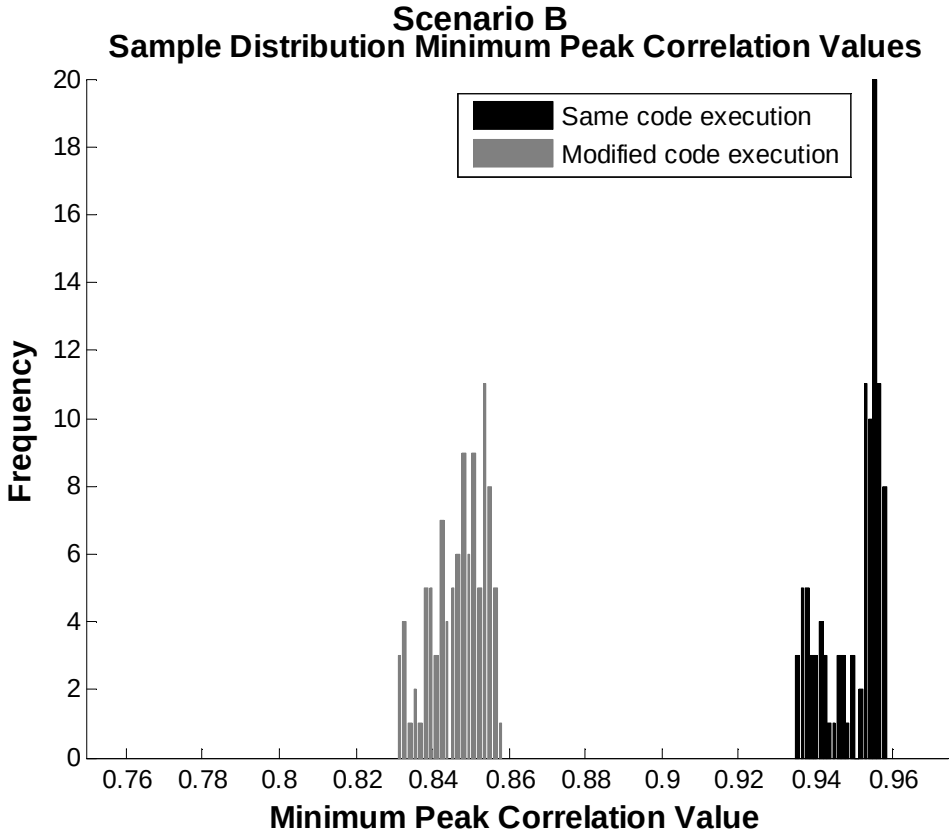

Figure 4.18: Sample Distribution Minimum Peak Cross-correlation Values Between Encrypted and Unencrypted Unicast Tx.

Source: C.R. Aguayo Gonzalez, Jeffrey H. Reed, "Detecting unauthorized software execution in SDR using power fingerprinting" [2], ©IEEE, 2010. Used with Permission.

slightly different instructions. In the figure, the trace labeled as alternative execution has a single bit transition less in that particular instruction than the base execution. It can be seen that, as a result, the base execution trace reaches a higher level at the peaks. During the rising slopes, however, there is very little difference between the two traces, hence little discriminatory information. In order to improve the performance of PFP and reduce processing requirements, it is necessary to concentrate on the sections of the traces (dimensions) that carry the most information (display a larger variance) and discard redundant information. Section 4.4.1 describes such process.

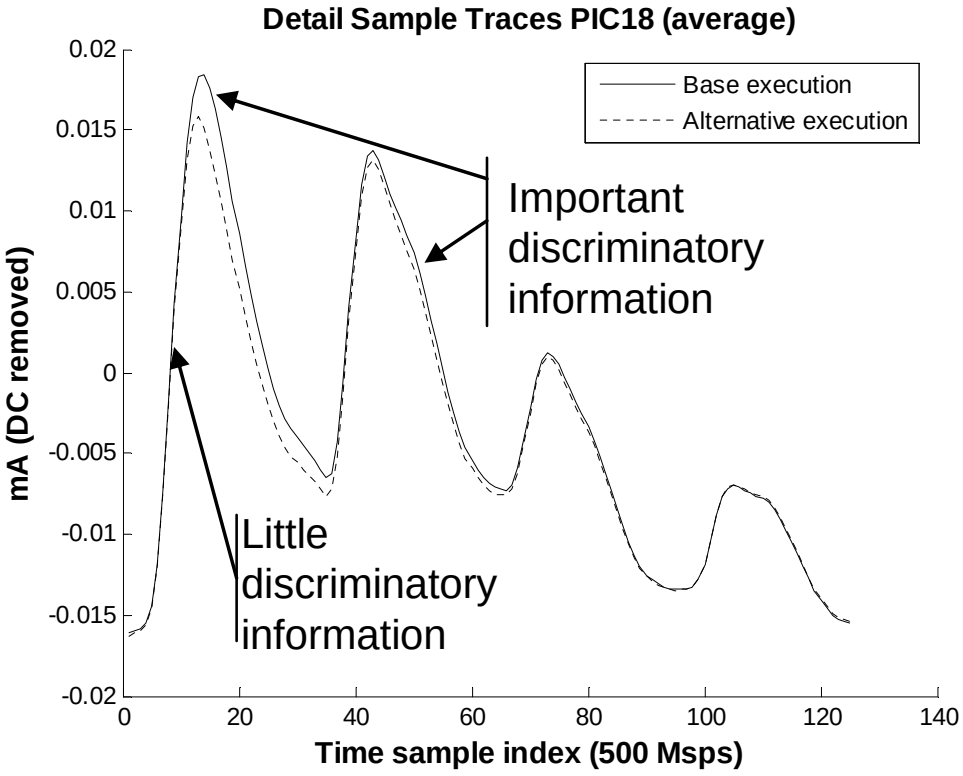

Figure 4.19: Sample trace detail showing different sections in a trace contain different levels of discriminatory information. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

There is, however, another aspect of trace variance that plays a critical role when characterizing a specific routine that takes random parameters as inputs. Executing with random parameters adds noise to signatures, reducing performance and increasing the probability of false alarm. For these cases, we would like to focus our attention to the dimensions that remain constant for during the execution of the target software, while ignoring the ones that add noise. This process is described in Section 4.5.

4.4.1 Platform Characterization Approach

In traditional pattern recognition systems, the process of selecting a subset of features that maximizes a specific criterion (in the case of PFP we want to maximize discriminatory information), is known as optimal feature selection. In clustering systems, this can be accomplished by projecting the traces to a transformed space with fewer dimensions by means of a linear transformation. This process is depicted in Fig. 4.20. The key for the success of this approach is to find the proper transformation that projects the traces from the most useful (or informational) perspective. In PFP, as depicted in Fig. 4.20, trace sections corresponding to a full clock cycle are reduced to a single point after the transformation. Classifiers are designed in the transformed space, reducing the number of dimensions that need to be considered during normal monitoring operation.

The transformation depicted in Fig. 4.20 is described by (4.10)

$$y = \mathbf{W}x \tag{4.10}$$

Where $\mathbf{W}$ is a carefully designed transformation matrix that maximizes a particular criteria of the traces in the transformed space and x is a column vector from $r_{\alpha}^{(i)}$ corresponding to a full clock or instruction cycle. Because we are trying to optimize discriminatory information, it is natural to follow an information theoretical approach. This optimization has been performed before and can be found in several sources in the pattern recognition literature, for example [ToG74].

In PFP, this platform characterization is performed under controlled conditions and is required only once per platform. We present two general approaches to identify the optimal transformation matrix: principal component analysis and linear discriminator analysis.

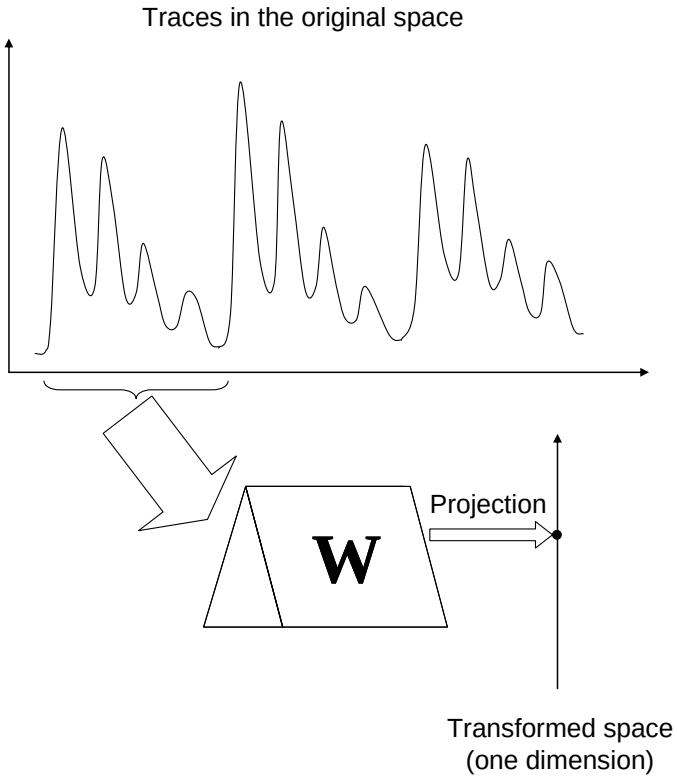

Figure 4.20: Platform characterization using a linear projection from the most informative perspective. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

4.4.2 Platform Characterization using Principal Component Analysis

A traditional approach to determine the appropriate **W** that optimizes the entropy (or information) in the traces is Principal Component Analysis (PCA). Assuming that the covariance matrices of the different classes[10], C_i, have identical Normal distributions $C_i = C$, the eigenvectors from C can be considered as the information bearers for the traces un-

[10]In this context, a class represents the power traces that result from the execution of a specific software

der consideration. Some of these eigenvectors carry more discriminatory information in the classification sense than others. Eigenvectors with small associated eigenvalue can be safely eliminated without losing too much information. It should be no surprise that the optimal transformation vectors are tied to these eigenvectors. The transformation matrix **W** is created by aggregating eigenvectors in descending order according to their corresponding eigenvalue. *Because in PFP we only need a single point per clock cycle, the transformation matrix* **W** *is given by the single eigenvector,* e_0*, of the covariance matrix associated with the largest eigenvalue, as shown in 4.11.*

$$\mathbf{W}_{\mathrm{PCA}} = e_0 \tag{4.11}$$

In order to characterize a specific processor and create a transformation matrix using PCA, it is necessary to observe the power consumption of the processor during several random clock cycles. The captured traces are aligned for every clock cycle to clearly show the sections of the traces that are affected the most by the dynamic behavior of processor execution. Once the traces are aligned, PCA is used to identify the transformation vector that accounts for the most variance in the traces.

Performing platform characterization using PCA is relatively easy to implement and well suited for complex platforms in which controlling the contents in the pipeline results too difficult.

4.4.3 Platform Characterization Using Linear Discriminant Analysis

PCA selects a feature subset that optimizes trace entropy. It does not consider, however, the specific differences between classes to select an optimal set of features that maximizes the distance between distributions. Linear Discriminant Analysis (LDA) maximizes the divergence between distributions. Divergence is a measure of distance between probability

distributions which is closely related to the concept of relative entropy in information theory. Using specific information from two classes, LDA identifies the optimal transformation matrix to project the traces from the unique perspective that yields the maximum separation between them. This is performed by choosing a transformation vector $\mathbf{W}_{\mathrm{LDA}}$ that is normal to the optimal discriminating hyper-plane between both distributions.

Following the assumption that traces are normally distributed, it can be shown [70] that the transformation matrix that yields a divergence extremum is given by the only eigenvector of $C^{-1}\delta\delta^T$ associated with a non-zero eigenvalue, where $\delta = \mathbf{m}_1 - \mathbf{m}_0$ and $\mathbf{m}_i$ is the mean vector of the class i. This vector is given by (4.12).

$$\mathbf{W}_{\mathrm{LDA}} = C^{-1}\left(\mathbf{m}_1 - \mathbf{m}_0\right) \tag{4.12}$$

$\mathbf{W}_{\mathrm{LDA}}$ provides the optimal projection to separate both classes. LDA can be extended to M discriminating classes. In this case, there will be $M-1$ eigenvectors associated with non-zero eigenvalues.

Performing platform characterization using LDA requires the development of two carefully tailored routines. These routines must execute specific instructions with specific addresses and parameters in the right sequence to create two sets of traces that show predetermined differences during a specific clock cycle. Training traces from the execution of both routines provide the two classes for which LDA will find the optimal discriminating hyperplane, which will in turn become the optimal transformation vector.

The LDA characterization routines need to properly load the pipeline such that in a specific clock cycle there is a known change during each execution stage (fetching, latching, execution, etc). The changes should be relatively small, preferably due to a few changing bits in the respective registers. The characterization routine is not unique, but it is platform specific as it depends on the architecture, instruction set, etc. of the specific platform. Different processors will likely require a different sequence.

Once traces from the execution of both sequences are captured and synchronized, Equation 4.12 is used to find $\mathbf{W}_{\text{LDA}}$. It is expected that platform characterization using LDA will provide the best performance, given the availability of two known classes, but its implementation is more complex than PCA.

4.4.4 Sensitivity to the Minimum Change Measurement Setup Description

For this experiment, we use the same setup described in section 4.3.3, with the only difference being that the RF daughter board was removed, and with it the charge pumps that introduce low-frequency interference. Therefore, there is no need for a high-pass filter for the traces in this experiment.

A routine is developed for this experiment with a dual purpose: 1) to introduce the smallest possible execution deviation to determine whether PFP can detect it and 2) to provide training routines to perform the platform characterization. We start by describing the first purpose, which also provides a baseline performance reference to compare the improvements of platform characterization. The test routine is shown in Listing 4.6 in which the contents of Register `W` are toggled from `00` to `0f` in an infinite loop using different instructions. This loop, starting in Line 22, represents our target code. The actual logic in the routine has no impact on the performance of PFP. The routine was chosen because it allows easy control of the number of bit transitions that happen during its execution. The results, however, do not depend on the specific software being executed and can be generalized to any arbitrary software execution.

```
0 BYTE i; //addr 00
  BYTE j; //addr 01
  BYTE k; //addr 10
  BYTE l; //addr 11

5 // Initialize the system
  BoardInit();

  // Initialize data variables
  _asm
```

```
10 movlw 0x07
   movwf i, 0    //addr 0x00
   movlw 0x0f
   movwf j, 0    //addr 0x01
   movlw 0x0f    //Set for minimum change
15 movwf k, 0    //addr 0x10
   movlw 0x1f
   movwf l, 0    //addr 0x11
   movlw 0x00
   _endasm
20
   //Target code infinite loop
   while(1){
       TMR0H = 0x00; //Restart TIM0
       TMR0L = 0x00;
25     LED_2 = 1;     //Trigger
       LED_2 = 0;

       _asm
           nop
30         iorwf  j, 0, 0  //w = 0f
           andlw  0x00     //w = 00
           movf   j, 0, 0  //w = 0f
           andlw  0x00     //w = 00
           movf   k, 0, 0  //w = 0f Change in k (one bit)
35         movlw  0x00     //w = 00
           xorwf  j, 0, 0  //w = 0f
           movlw  0x00     //w = 00
           iorwf  j, 0, 0  //w = 0f
           xorlw  0x00     //w = 00
40         nop
           ... x 10
           nop
       _endasm
   }
```

Listing 4.6: Software routine used for platform characterization and minimum sensitivity evaluation

The routine shown in Listing 4.6 represents the **base execution**. Preceding the target code, we create a trigger using an LED. The "`NOP`" instruction between the trigger and the target code works as a buffer to isolate the target traces form any residual effects from the trigger. Inside the main loop, Register `W` is toggled from `00` to `0f` creating four bit transitions in that register every instruction. The **alternative, or modified, execution** is designed to have one less bit transition. In Line 14, we change the contents of `k` from `0f` to `07`. This way, when the target code is executing Line 34, the parameter `k` is loaded onto Register `W`, which goes from `00` to `07`, with only three bits switching in that instruction. Note that there is just one

bit difference between this modified code and the base execution which loads Register `W` with a `0f`. Everything else in the execution is kept the same, including instructions, parameters, and addresses. This one-bit change actually affects two clock cycles, as there is one less transition coming into that instruction and one less coming out of it. Trailing the target code there is a string of "`NOP`" instructions before the loop is repeated.

A detail of a typical power trace from one full execution cycle of our target code is shown in Fig. 4.21. The effects of the trigger on the power traces are clearly visible as two square steps. Individual instruction cycles can be identified as groups of four spikes that repeat every 125 samples. Using timing information from the processors documentation, we can determine the section of the trace that corresponds to the execution of the target code. In Fig. 4.21, this section is highlighted as a solid line that spans ten instruction cycles. The target code consists of ten assembly instructions, each taking one bus cycle to execute.

Several traces captured from the base and alternative executions are averaged together to provide a clean picture of the total effect of one less bit transition. The averaged traces are shown in Fig. 4.22, where ten clock cycles corresponding to the execution of the target code are visible. The average from several base execution traces represents the base signature. Around sample index 650, a small difference between both traces can be seen. The difference is more noticeable in the top of Fig. 4.22, which provides a closer look. Along with the similarity of the traces from both scenarios, it is also evident that traces are largely correlated due to over sampling and also that only certain sections of the traces carry useful discriminatory information.

As a reference, we provide the results of a naive classification approach in the time domain without platform pre-characterization. In this approach, each captured trace of length L = 1250 (the length of the target code) represents a point in a L-dimensional Euclidean space[11]. The Euclidean distance is taken from the base signature to each incoming test trace. Evaluation traces are different from the training traces used to obtain the base signature.

[11]Note that we are using a different feature selection approach that the one described in Section 4.3.4

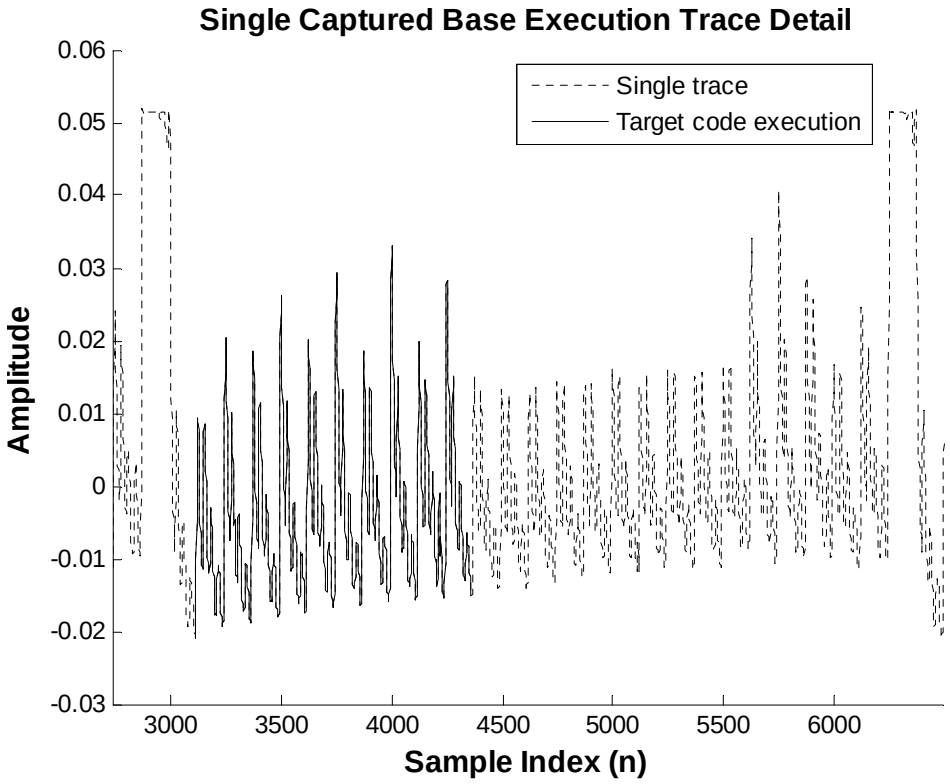

Figure 4.21: Sample trace from baseline code execution to evaluate ability to detect minimum power consumption change. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

The Euclidean distance from traces from both routines to the base signature have the distributions shown in Fig. 4.23. In this naive example, the classification performance of PFP is not encouraging, as there is barely any difference between the distributions from both executions. This poor performance is expected considering the small differences in power consumption between the base and alternative scenarios.

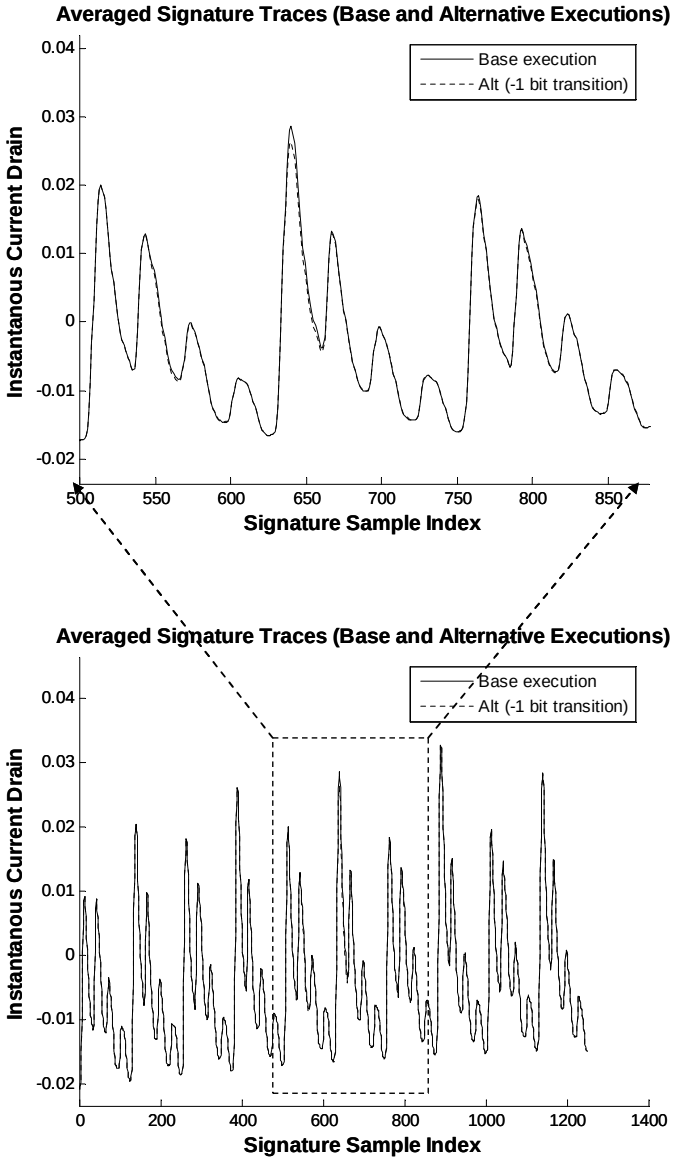

Figure 4.22: Baseline code execution average signature. Each point represents a projection on an n-dimensional Euclidean space. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

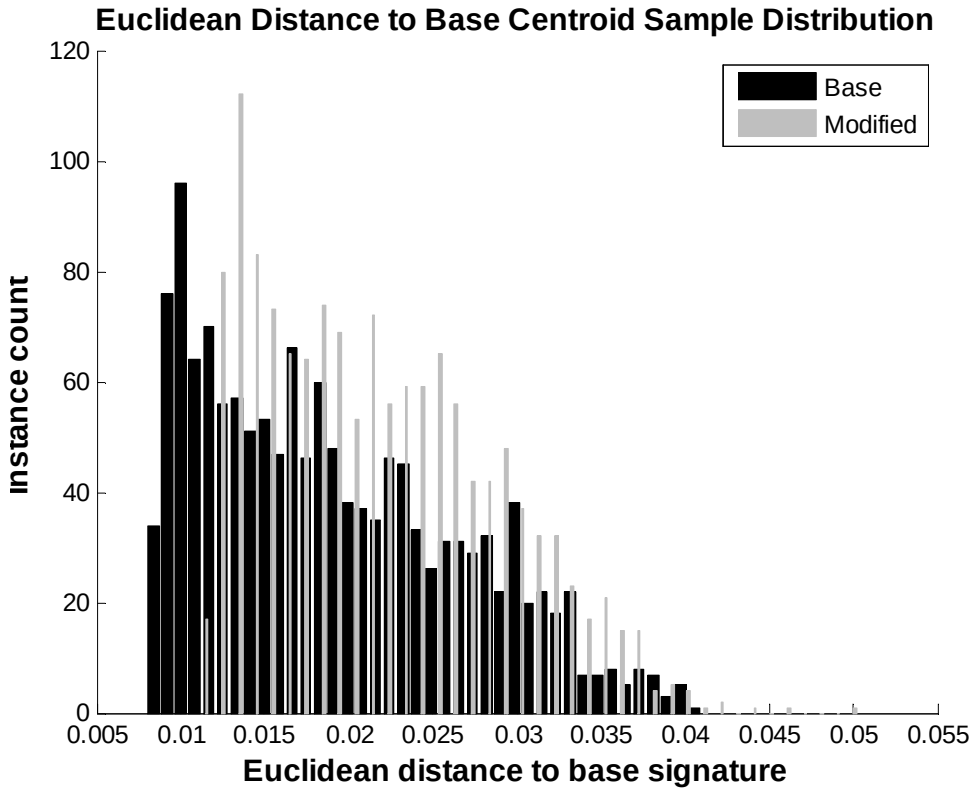

Figure 4.23: Sample Distribution of Euclidean distances from the average signature extracted from the execution of the baseline code. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

4.4.5 Improving PFP Performance Using PCA Platform Characterization

The first platform characterization is performed by applying PCA. For this process we use all clock cycles corresponding to the execution of our target code in the routine shown in Listing 4.6. The trace containing the full execution of the target code is divided into different sections corresponding to single clock cycles. The subsections are then aligned and PCA is

used to find the transformation vector $\mathbf{W}_{\text{LDA}}$ corresponding to the eigenvector that accounts for the most variance. The result of this transformation in both the base and modified executions is shown in Fig. 4.24. Notice how the oversampled trace is reduced to a single point for each instruction cycle, whose value is roughly related to the total power consumed in that cycle.

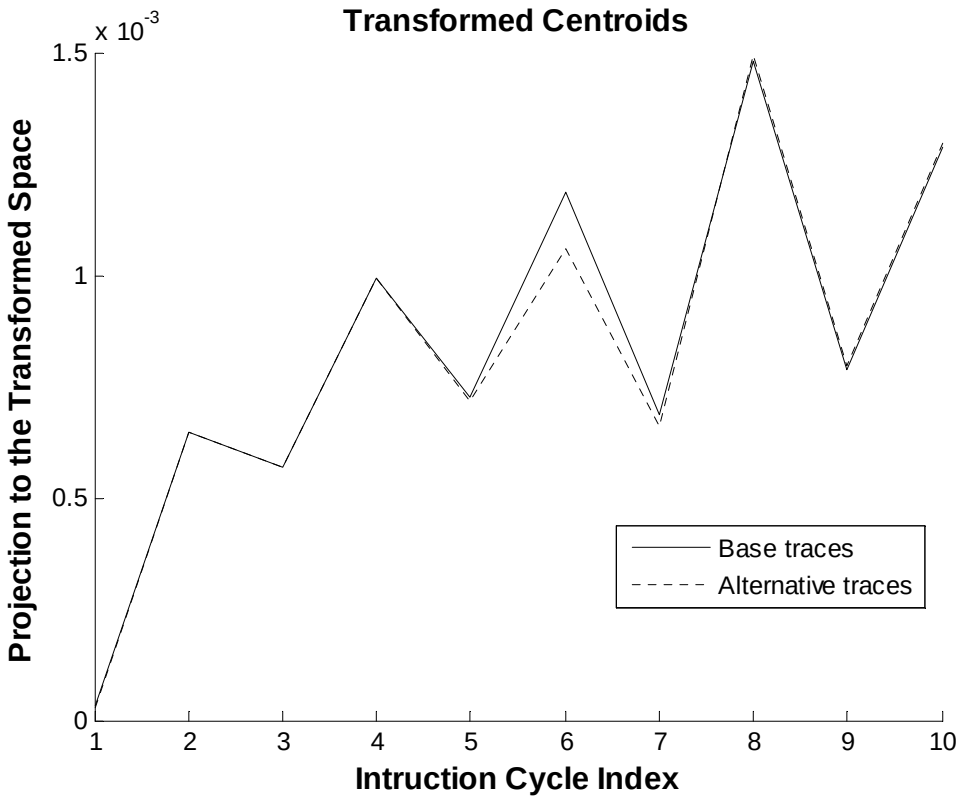

Figure 4.24: Centroids of the transformed traces for the base and alternative executions. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

After performing the transformation, the test traces from the evaluation routine are processed again to demonstrate the performance improvements of platform pre-characterization. Same

as the naive example shown before, every sample point in the transformed trace corresponds to a dimension in a transformed Euclidean space. The minimum distance distributions from the test traces to the signature, both in the PCA transformed space, are shown in Fig. 4.25.

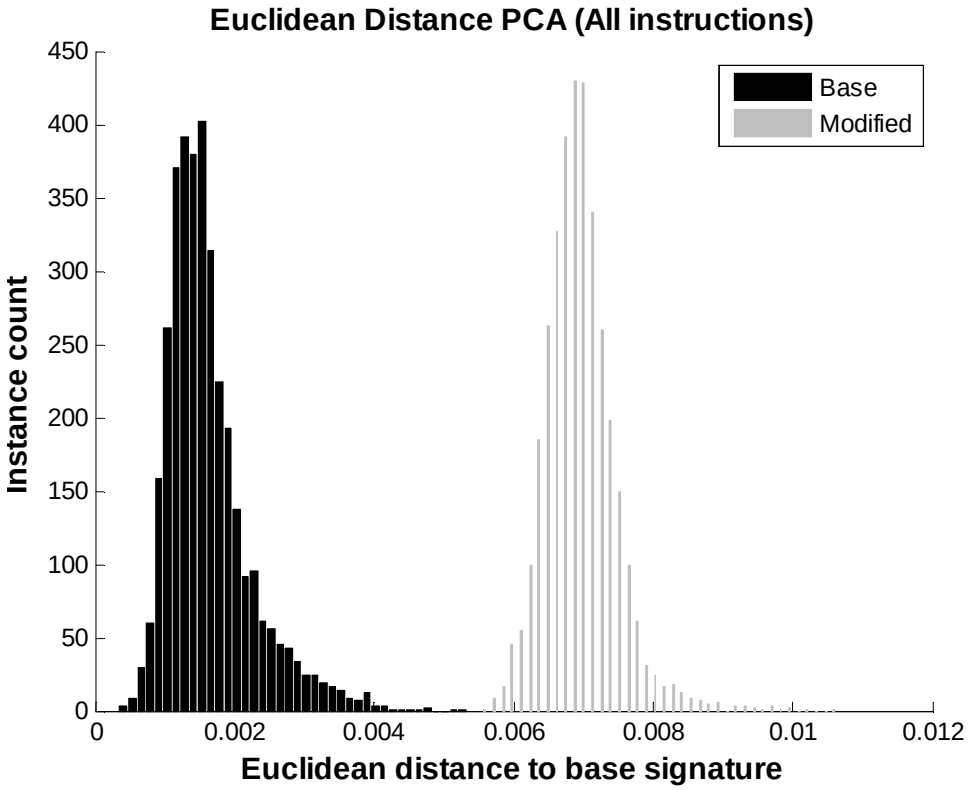

Figure 4.25: Sample distribution of Euclidean distances from the baseline signature in the transformed space obtained using PCA. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

When comparing Fig. 4.23 with Fig. 4.25, a clear separation is seen between the bulks of the distributions, which represents a clear improvement with respect to the naive classification performance.

4.4.6 Improving PFP Performance Using LDA Platform Characterization

In order to obtain the training traces necessary to apply LDA, we execute the base routine and a slightly modified version. We obtain the special platform characterization traces by comparing two sets of traces: from the base execution, which is once again the code in Listing 4.6 and a slightly modified version shown in Listing 4.7. The changes in execution are carefully selected to cause one less bit transition on each execution stage compared to the base execution. In this modified version, the instruction in Line 36 is changed from `xorwf` which has the opcode `0001 10da` to `iorwf` with opcode `0001 00da` (the optional arguments `d` and `a`, control the destination and RAM access bit, respectively, and are kept with the same value in both versions). During execution the difference in the opcodes will cause one-less bit transition when latching the instruction word. The parameter in the instruction changed from `j`, located at address `0x01`, to `i`, located at address `0x00` in Access RAM. Once again, the change will create one-less bit transition when executed. Furthermore, notice that the contents of `j` and `i` also differ in one bit. This will also translate into one less bit transition when parsing the parameter, when executing the instruction and when writing the results.

```
    ...
35  movlw   0x00      //w = 00
    iorwf   i, 0, 0   //w = 07
    movlw   0x00      //w = 00
    ...
```

Listing 4.7: Modified routine for LDA platform characterization

For platform characterization we use only traces corresponding to the execution of Line 36. The average of these traces is shown in Fig. 4.26.

Using these traces, we perform LDA to identify the optimal discriminating hyperplane and the linear transformation vector $\mathbf{W}_{\text{LDA}}$ that projects our traces from the most informative perspective. The test traces from the evaluation routine are processed again to demonstrate the performance improvements of platform pre-characterization. The minimum distance

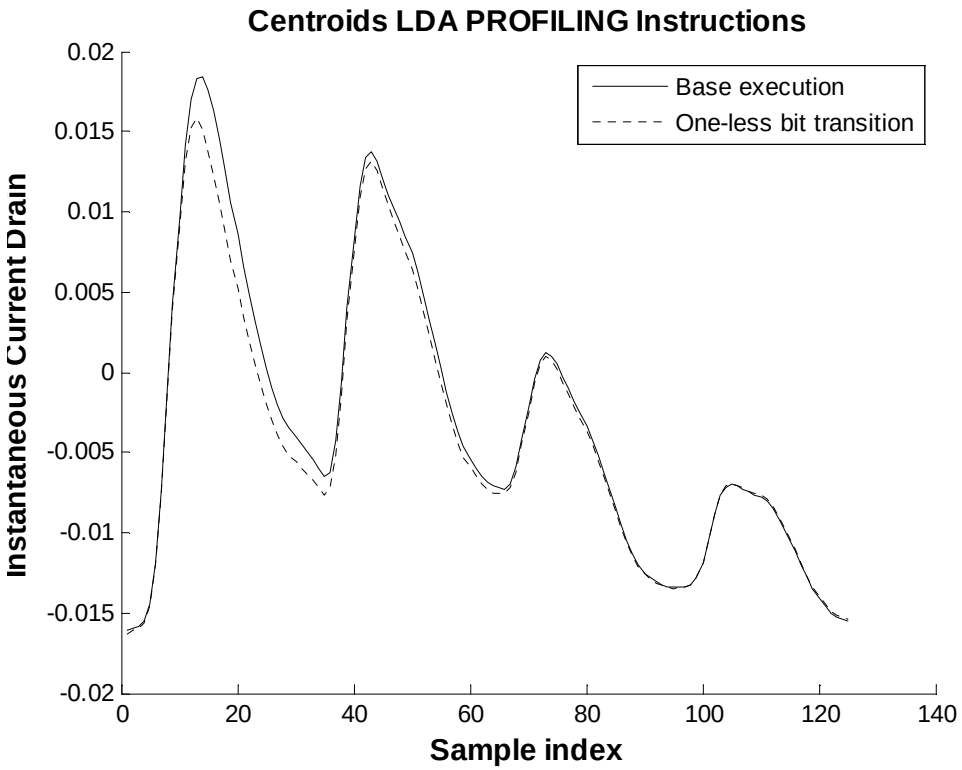

Figure 4.26: Centroids of traces from profiling instructions for LDA. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

distributions from the transformed test traces to the signature in the new LDA transformed space are shown in Fig. 4.27.

LDA provides the best performance, given the availability of two known classes, when compared to PCA for platforms characterization. The PCA approach, however, is easier to implement and provides an alternative for complex platforms for which controlling the contents in the pipeline results too difficult.

The resulting sensitivity from the different approaches to perform platform characterization can be seen in Figure 4.28, which shows a Receiver Operating Characteristic (ROC) curve

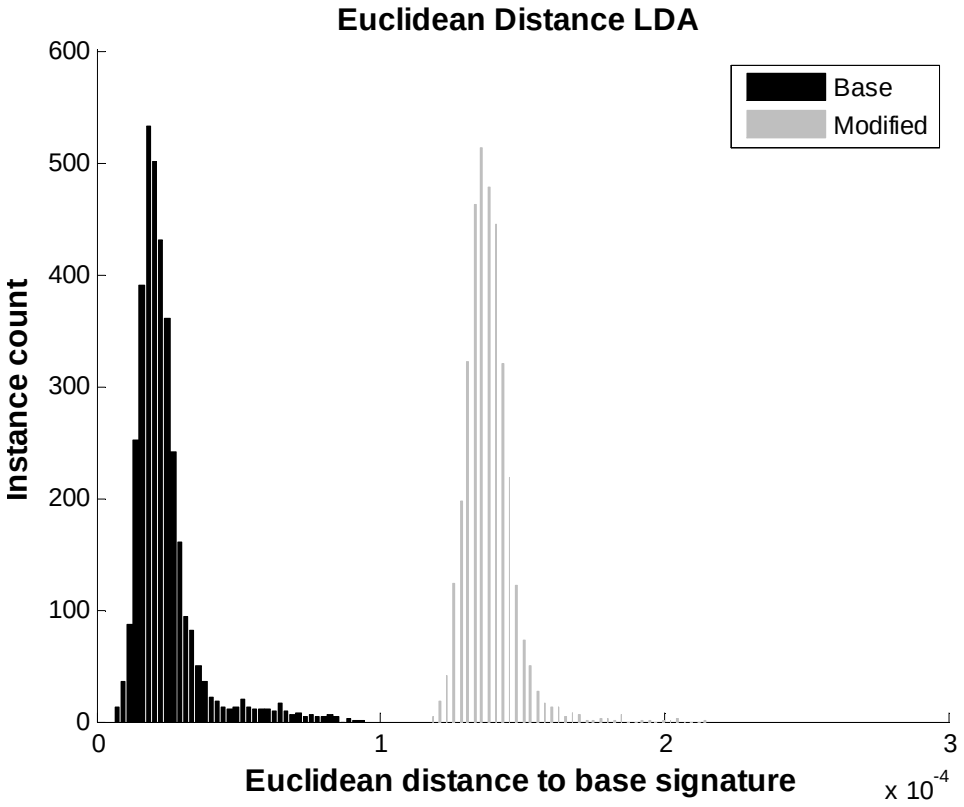

Figure 4.27: Sample distribution of Euclidean distances from the baseline signature in the transformed space obtained using LDA. *Source*: Used with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4]. ©Springer Science+Business Media, LLC 2011.

for both PCA and LDA characterizations.

In the figure, the observed ROC from both characterizations is plotted along with the expected results from distribution fitting of the observed distributions. In both cases, PCA and LDA, the observed distributions from the base execution are fitted to a Generalized Extreme Value distribution, while the distributions from the modified executions are fitted to a Normal distribution. The sensitivity performance can be better appreciated in the

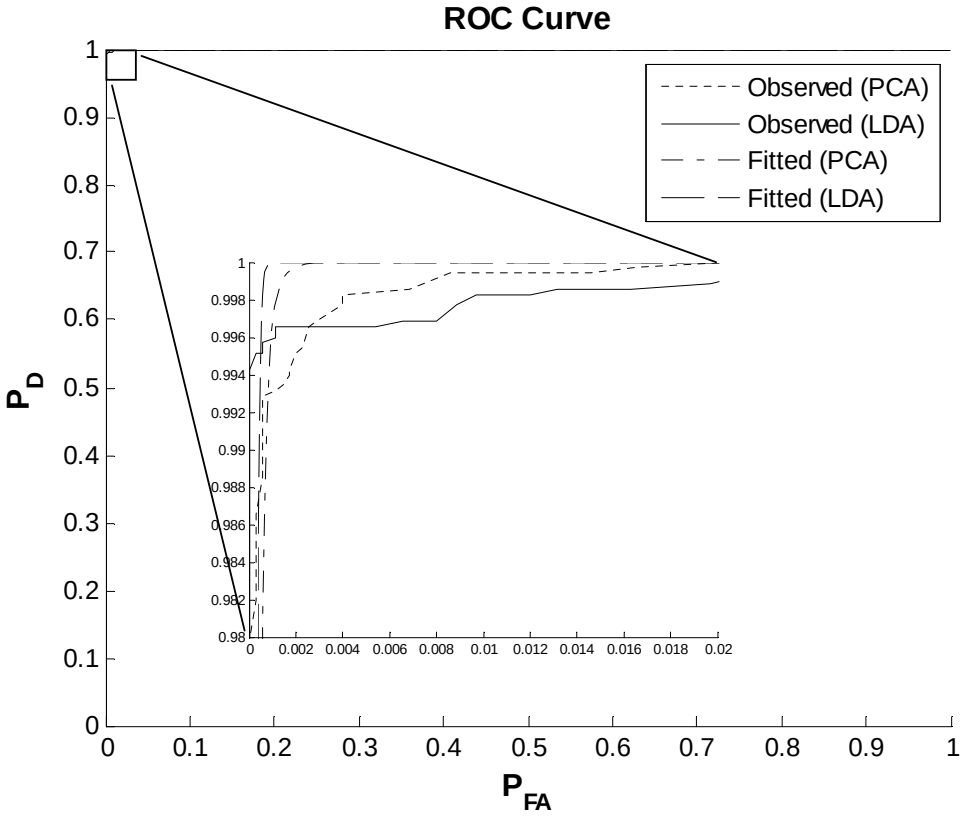

Figure 4.28: ROC curve to depict the minimum sensitivity achieved as a result of different platform characterization approaches

detailed plot. The observed sensitivity for both approaches is very close. LDA is able to achieve a very low P_{FA} of 0.001 for a P_D of over 0.996. This approach, however, has to pay a significant performance penalty to improve the observed P_D. For a section of the curve with $P_D > 0.997$ PCA has a slightly better performance. For the ROC curve using the respective fitted distributions, LDA has a better, overall performance than PCA.

4.5 PFP Monitoring of Routines with Random Parameters[12]

When a routine executes with random parameters, it is necessary to remove the added variance from the PFP signatures in order to prevent performance degradation. One approach is to use PCA to identify a linear transformation similar to the one described for platform characterization. The linear transformation takes place on the transformed space (after the oversampled traces have been reduced to a single point in a transformed Euclidean space) and aims to remove the features responsible for the most variance. The linear transformation is performed on the complete signature using a transformation matrix $\hat{\mathbf{W}}$ constructed by applying PCA and removing the eigenvectors of the covariance matrix with the highest eigenvalues. This is the exact opposite from the approach used for platform characterization. It is important to note, however, that a potential attacker could learn the dimensions (sections of the traces) that are affected by random parameters and hide the attacks there. Although, the potential scope and impact of the attack becomes severely restricted.

There may be some difficult cases in which it is not possible to remove the dimensions that account for the most variance while still maintaining sufficient discriminatory information. **For these cases, it is still possible to apply PFP by performing the characterization and assessment processes using a predetermined input.** Using a predetermined input facilitates characterization by improving execution determinism, but also opens the possibility of a potential attacker learning the specific input used and hiding in its presence. The key for PFP is that even in this case the attacker has to read the inputs and, due to observer's effect, would disrupt the signatures and disclose his presence.

[12]The material in this section is presented here with kind permission from Springer Science+Business Media: Analog Integrated Circuits and Signal Processing, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance", 69(2), 2011, 307-327, C.R. Aguayo Gonzalez and J.H. Reed [4].

4.6 Conclusions

This chapter described the general operation of PFP on deterministic computing platforms. It also presented the results of feasibility experiments for PFP in a PICDEM Z commercial radio platform. The results demonstrate the feasibility of PFP integrity assessment monitoring in SDR commercial platforms for regulatory compliance and intrusion detection. In the first experiment, a PFP monitor is able to detect when unauthorized code is inserted to change the operational mode of the RF transceiver to Turbo Mode, which impacts the spectral emissions. Furthermore, in a more challenging scenario, the monitor is also able to detect a similar violation during an explicit configuration of the RF module, which involves only a parameter change to go between Normal and Turbo modes. In both scenarios, the monitor is able to detect, with a 100% accuracy in 200 trials, modifications in the configuration code that can lead to deviations in spectral emissions.

In a second experiment, we successfully demonstrated the ability of PFP to accurately discriminate between two types of transmission involving different encryption options. In the first scenario, the PFP monitor is able to effectively discriminate between the execution of an encrypted unicast transmission and an unencrypted broadcast transmission. More importantly, in a second scenario the monitor is able to discriminate, with 100% accuracy in 200 trials, between encrypted and unencrypted unicast transmissions.

In this chapter, we also described a technique to improve the performance of PFP by pre-characterizing the way a specific platform consumes power. This pre-characterization is performed by designing a transformation matrix that projects the power traces from the most informative perspective. The linear transformation allows PFP to emphasize the sections of the traces that carry the most discriminatory information while ignoring redundant information. The transformation matrix is designed using two approaches: PCA and LDA. The latter yields a better performance, but requires the development of a carefully crafted routine that provides bit transitions in every execution stage in a single clock cycle. The former is much simpler to implement, as it only requires observing a large number of random

clock cycles, but it yields a slightly reduced performance. In both cases, however, the PFP monitor was able to successfully discriminate between two executions that differ in only a single bit transition (the smallest possible disruption).

Chapter 5

PFP in non-deterministic platforms

In the preceding chapters, we introduced the general theory of power fingerprinting (PFP). We also presented experimental results that demonstrate the ability of PFP to monitor basic, deterministic platforms. In these results, PFP was able to detect execution deviations that affected security settings and spectral emissions of a commercial radio platform. PFP was also able to detect the minimum change possible of one bit due to a logic execution deviation by pre-characterizing the platforms unique power consumption characteristics. These feasibility results, however, assume deterministic execution, independent of the specific state sequence of the processor. This condition is commonly met in basic processors for low-power applications, but the same cannot be said about more complex platforms with sophisticated architectures and non-deterministic behaviors. Modern high-performance processors include a variety of dynamic elements designed to improve the execution speed. Unfortunately, these dynamic elements link the current processor operation to previous execution states.

This chapter takes PFP a significant step forward. It describes the general characteristics of more complex, non-deterministic platforms and the impact they have on PFP. In this chapter we also present further feasibility results on the ability of PFP to monitor the execution integrity of complex software on a high-performance Android platform used for mobile communications and computing systems. The feasibility results show PFP detecting execution

deviations similar to those created by covert intrusions that trigger special functionality only under specific conditions. Another feasibility experiment on the Android platform shows the ability of PFP to detect a real privilege escalation attack (jaibreaking). The malicious attack is known as "rageagainstthecage" and is one of the exploits used in the famous "DroidDream" malware, that was distributed in malicious apps from the official Android Market in early 2010. In these results, the different requirements for synchronization and feature extraction are highlighted. Throughout the chapter, different techniques and processes are developed for feature extraction and synchronization signaling, which establish a general architecture for integrating PFP into non-deterministic platforms.

5.1 Non-deterministic Computing Platforms

In the context of PFP, we refer as a non-deterministic computing platform to a processor that includes modern dynamic features to improve execution performance. Such dynamic features introduce uncertainty about the exact execution status and timing. The uncertainty is introduced because the behavior of dynamic modules adapts based on previous states. Such dynamic features are usually reserved to modern high-performance devices because of the extra circuitry necessary to implement the features increases complexity, cost, and power consumption.

Modern processors contain deep pipelines to boost performance and execute more tasks in a single clock cycle.

A consequence of deeper pipelines is an increased performance penalty when a conditional instruction is executed. The processor can stall until the result of the condition is known and the next instruction is determined or it can guess the result and load the pipeline with the instructions that correspond to that guess. If the result of the condition requires a different path than the one predicted, it is necessary to flush the contents of the processor. In order to reduce missed prediction penalties modern processors implement a technique

called dynamic branch prediction, which tries to guess the results of a conditional branch based on execution history. From the PFP perspective, dynamic branch prediction changes the timing of the signatures and can insert extra states that impact the look of the signatures. While a modern processor will ultimately execute the same logic every time it encounters the same code, the sequence of states that takes to achieve such result can be different. In other words, the processor takes a different route to reach the same logic state. Furthermore, a deeper pipeline spreads the impact of random parameters across more instruction cycles

Besides deep pipelines and dynamic branch prediction, there are several other features in complex processors that can impact PFP signatures. For instance, cache memory can add execution uncertainty if missed. Also, modern processors tend to have multiple execution units and, while scheduling tends to be reliable, there is the possibility that some instructions get assigned differently in different execution instances.Table 5.1 lists some examples of non-deterministic computer platforms.

Table 5.1: Examples of non-deterministic platforms from the PFP perspective

Popular commercial non-deterministic platforms
ARM's Cortex family [12]
IBM's PowerPC 7xx processors
Freescale's PowerQUICC
Intel's Atom family of processors

In order to exemplify why modern, high-performance processors are considered non-deterministic, it is enough to cite the ARM Architecture Reference Manual [13]. The manual, in its detailed description of the NOP instruction, states: "*The timing effects of including a NOP instruction in code are not guaranteed. It can increase execution time, leave it unchanged, or even reduce it.*"

Besides the inherent hardware uncertainties, modern operating systems normally deployed

with high-performance processors also introduce a significant amount of execution uncertainty. They main concern, from the PFP perspective, is that most of them require a context switch to get into supervisor mode, which relies on a software interrupt. This approach makes accessing Kernel resources from the User space subject to scheduler status.

5.2 General PFP Operation on Non-Deterministic Platforms

As explained in the previous section, non-deterministic features can impact the state sequence necessary to accomplish a specific operation. Non-deterministic operation affects timing and alters intermediate states. These uncertainties force the selection of discriminatory features in PFP different from those used in deterministic platforms.

From the PFP perspective, it is more difficult to use time-domain discriminatory features. The selected features must be robust against such uncertainties or the PFP system will have to compensate for them. Another option is for the characterization process to extract multiple signatures from the most likely scenarios (i.e. characterize when the branch prediction is correct and also when it is missed). Depending on the complexity of the code, this last option may be too complicated and result unfeasible.

Fortunately, there are relatively simple feature extraction techniques that allow capturing the nature of the internal transitions while being relatively robust to specific timing changes. For example, power spectral analysis, in the frequency domain, is capable of capturing most of the information carried in the logic transitions, without being significantly disrupted by minor timing changes. This type of feature extraction technique is used in the feasibility experiments presented next in this chapter.

Another important challenge involves timing uncertainties due to indirect access to Kernel resources. As mentioned before, these timing uncertainties happen because the scheduler

may schedule different tasks before servicing a user-space request. Timing uncertainties can have a negative impact on PFP characterization and monitoring when such resources are required to provide synchronization signaling from the User space. This is not an exhaustive list of issues and challenges in applying PFP monitoring to sophisticated, non-deterministic platforms. The specific characteristics of a given platform may present unique challenges to PFP.

5.3 Case Study: Android Platforms

Due to the wide variety of non-deterministic computing platforms, it is not possible to generalize specific techniques that guarantee successful PFP monitoring. As mentioned before, each platform will present unique challenges. For instance, the discriminatory features selected will be different. The techniques and approaches necessary to properly characterize and monitor execution will also be different, as will be the sensitivity requirements for each system.

For this work, a case study is used to introduce PFP to non-deterministic platforms and demonstrate its feasibility. The results provide a reference for future deployments and offer valuable guidelines on how to deal with expected issues. For this case study, we selected the open-source Android platform, a popular framework widely deployed in a variety of smart phones and tablets, and which has been targeted by malicious attacks. The following section describes the Android framework in more detail. It also describes the processing platform used to perform the feasibility experiments.

5.3.1 Android Framework

Android is a software stack for mobile devices which includes an operating system, middleware, and key applications. Android is an open source project managed by the Open

Handset Alliance [6] led by Google. Android provides a framework for developers to access device hardware, location information, background services, and much more. There is also an open-source development platform that supports a large community of third party developers. Android is the leader operating system in smart-phones by market share at the time of this writing, with several cell-phone manufacturers using it in their handsets, including Samsung, HTC, and Motorola.

The Android software stack and architecture is shown in Figure 5.1.

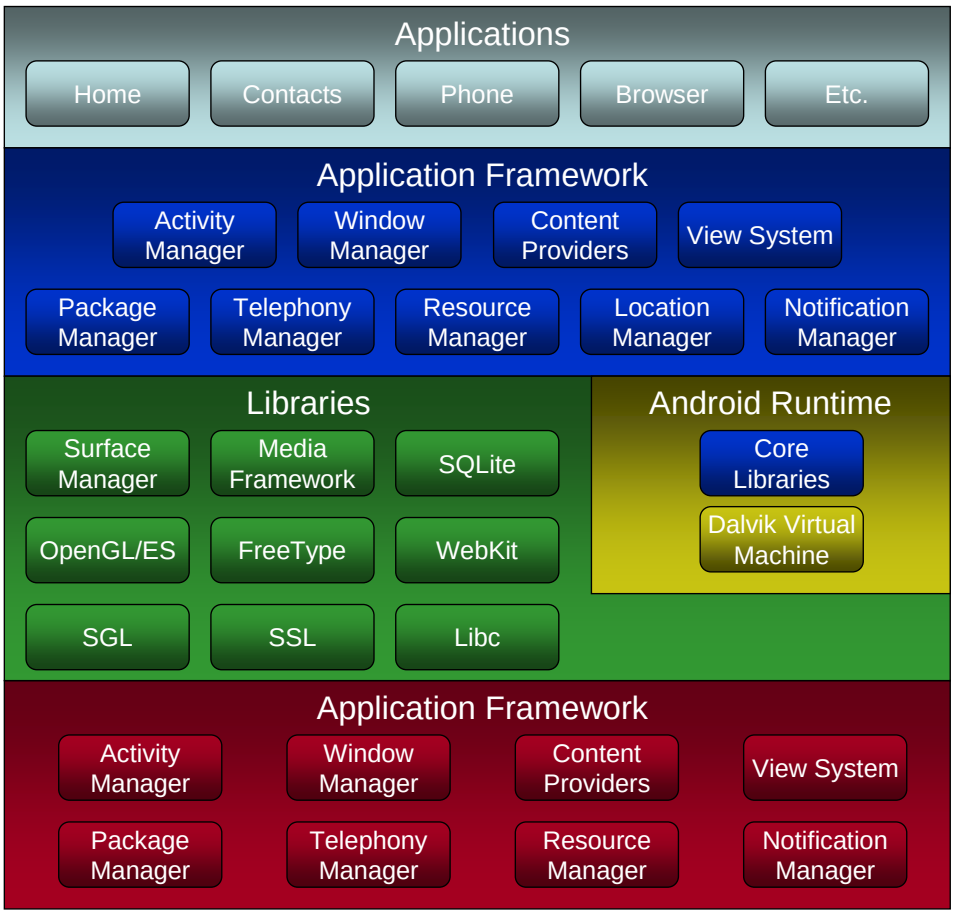

Figure 5.1: Android Architecture.
Adapted from Android Dev Guide [10]. Used under fair use guidelines, 2011.

The Android applications, also known as apps, reside at the top of the software stack. All the

apps are written in the Java language. Android itself ships with a set of core applications readily available. Underneath the application layer there is a set of services and systems for content and management, including: an extensible set of "Views", (e.g. lists, grids, text boxes, buttons, and an embeddable web browser) used to build applications; "Content Providers" that enable applications to access data from other applications; a "Resource Manager" that provides access to non-code resources; a "Notification Manager" that allow applications to display alerts; and an "Activity Manager" that manages the lifecycle of applications and provides navigation tools. There are other managers in this layer that allow access and control of different features and peripherals in the device.

Android provides custom versions of support libraries used by various components in the frameworks and which are also available for applications. The available libraries include: a BSD-derived implementation of the standard C system library, tuned for embedded Linux-based devices; a modern web browser engine; 3D (with hardware accelerator support when available) and 2D libraries; font rendering; a relational database engine; and other media libraries.

Being based on Java, the Android framework also implements is own virtual machine called Dalvik. Every Android application runs in its own process, with its own Dalvik instance. Dalvik was written to allow a device to run multiple VMs efficiently and executes files in the special Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based and runs classes compiled by a Java language compiler that have been transformed into the .dex format. As part of Android Runtime, there is a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. The Dalvik VM relies on the Linux kernel.

More information about Android can be found in [8]

5.3.2 The Linux Kernel

Android is based on the Linux Kernel 2.6, although the Android kernel is maintained as a separate branch, to provide core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack. The main components of the Linux Kernel are shown in Figure 5.2.

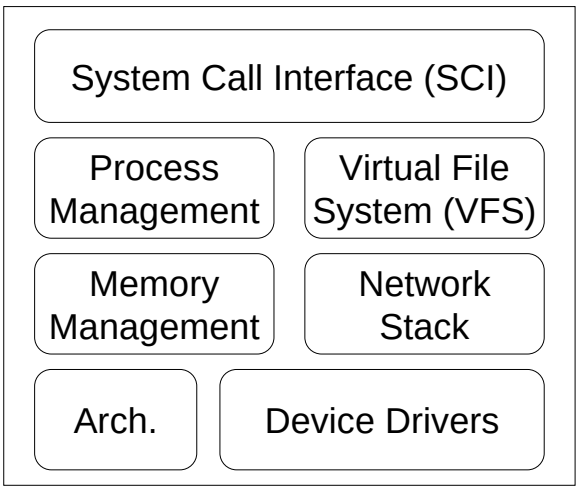

Figure 5.2: Main components in the Linux Kernel

The Linux Kernel, commonly used in embedded platforms [34], has a monolithic architecture that supports preemptive multitasking, virtual memory, shared libraries, demand loading, memory management, and threading. Unlike traditional monolithic kernels, however, the Linux kernel provides full access to the hardware using loadable device drivers, or modules, that can be loaded or unloaded while the systems is running. Device drivers and other kernel extensions operate in the Kernel Space and require a system call to serve as an interface to a system request generated in the User space.

5.3.3 Android Security Model

The Android OS provides several key security features to protect user data, protect system resources, and isolate applications. These features include [9]:

- Robust security at the OS level through the Linux kernel
- Mandatory application sandbox for all applications
- Secure inter-process communication
- Application signing
- Application-defined and user-granted permissions

As the foundation of Android, the Linux Kernel provides a user-based permissions model, process isolation, and an extensible mechanism for secure IPC. The user-based security model of the Linux Kernel prevents different users from accessing each other's data as well as exhausting each other's memory, processing resources, or devices. The Android platform takes advantage of the Linux user-based protection philosophy and assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process. This approach sets up an application "sandbox" in which applications cannot interact with each other and have limited access to the operating system. Since the application sandbox is implemented by the kernel, this security model extends to native code and to operating system applications. All of the modules in Figure 5.1 that reside above the kernel (including operating system libraries, application framework, application runtime, and all applications) run within a unique sandbox.

5.3.4 Test Platform Description: BeagleBoard

For the feasibility experiments in this chapter, we selected a research platform called BeagleBoard (rev. C4), shown in Figure 5.3. The BeagleBoard is a low-power, low-cost, open-source

single-board development platform based on the popular OMAP3 processor from Texas Instruments [OMAP]. The OMAP3 platform contains superscalar ARM Cortex-A8 core and TI's C64x+ DSP. The BeagleBoard is a representative example of non-deterministic platforms because the OMAP3 family is used in several commercial products, including: Motorola Droid, Palm Pre, Samsung i8910, and Nokia N900.

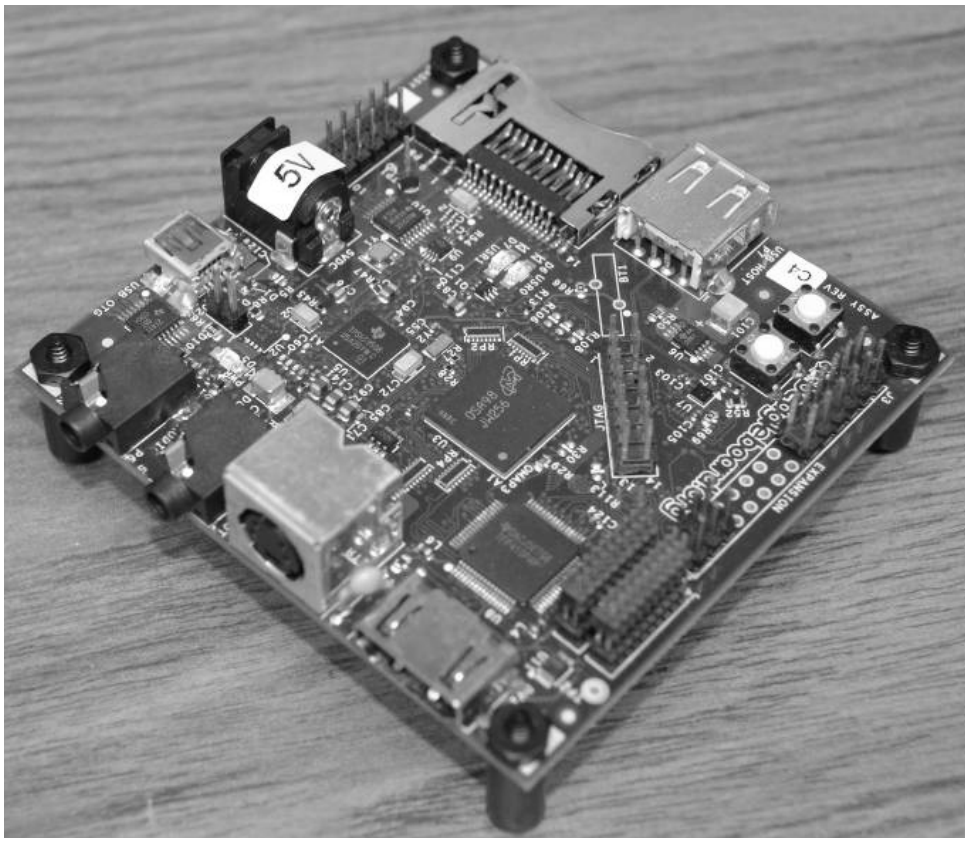

Figure 5.3: BeagleBoard Development Platform

The BeagleBoard is designed specifically to address the Open Source Community and was equipped with a minimum set of features to allow the evaluation of the OMAP3530 processor. This is a low-cost platform, it is not intended as a full development platform, as many of the features and interfaces in the OMAP3530 are not supported [15]. The BeagleBoard is sold to the public under the Creative Commons SharedAlike license, has a large community of developers, and is supported by several open-source projects including Android, Angstrom

Linux, Fedora, Ubuntu, etc.

The Android software stack is available open-source by the OpenHandset Alliance and the BeagleBoard is a popular architecture for porting among hobbyists. Therefore, there are several open-source Android ports that support it, including RowBoat [60], which is the Android version used in this research.

OMAP3

The processor featured in the BeagleBoard is the OMAP3530 high-performance applications processor. This processor is based on the enhanced OMAP 3 architecture, designed to provide video, image, and graphics processing commonly used in streaming video, mobile gaming, video conferencing, and high-resolution still image applications, among others [37]. The OMAP3 is intended for mobile applications and includes several power-management techniques to reduce power consumption. The OMAP3 processor is a complex system with several subsystems, co-processors, and hardware accelerators, as shown in Figure 5.4.

From the PFP perspective, the main modules in the OMAP3 core processing engines include:

- Microprocessor unit (MPU) subsystem based on the ARM Cortex-A8 microprocessor
- High-Performance image, video, audio (IVA2.2) subsystem with a C64x+ digital signal processor (DSP) core

Besides the core processing modules, the OMAP3 includes hardware support for 3D graphics acceleration (POWERVR SGX), camera image signal processing, display systems, memory controllers, and many other peripherals. For this chapter's feasibility experiments, only the ARM MPU section is utilized.

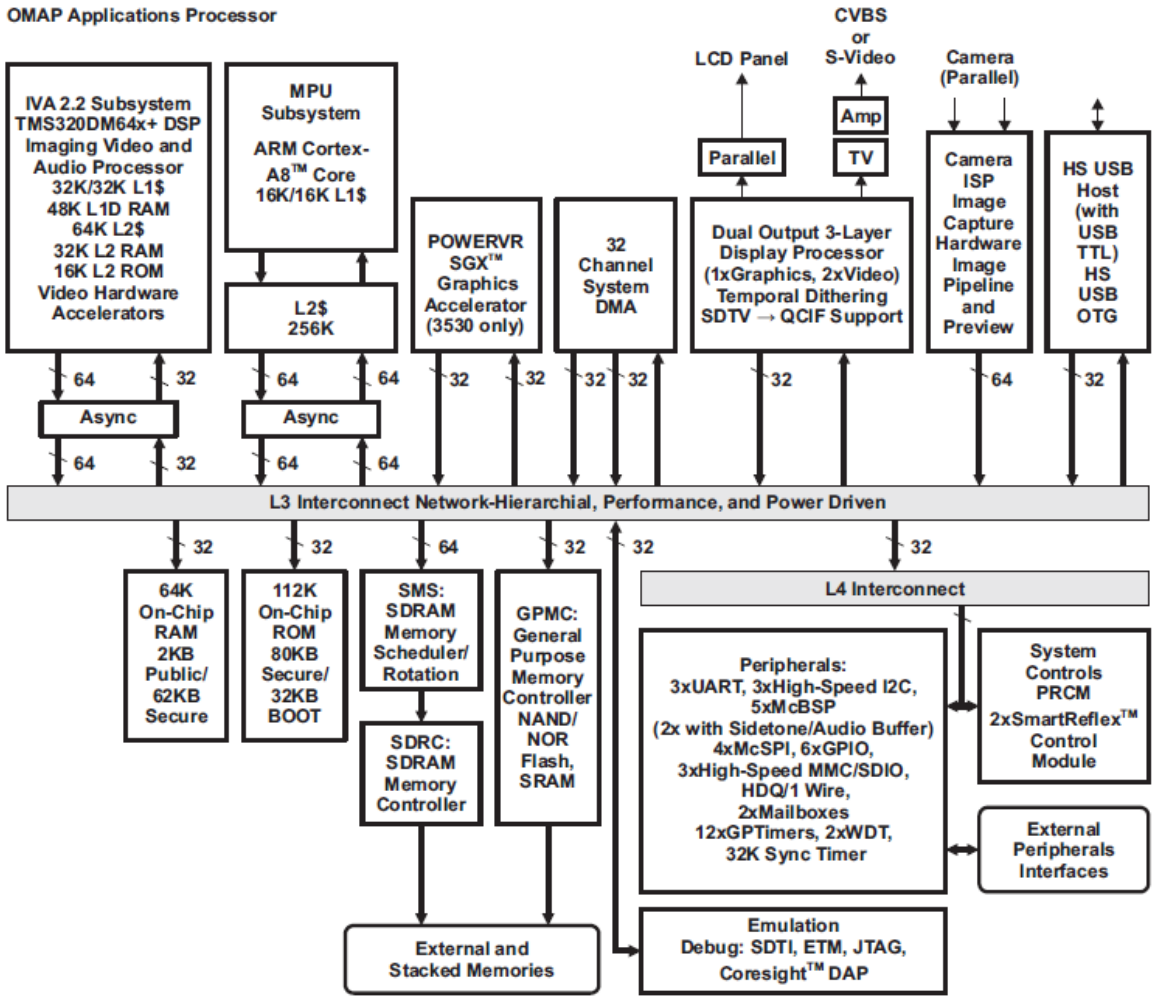

Figure 5.4: OMAP3 Functional Block Diagram [37]. Courtesy Texas Instruments

ARM Cortex-8

The ARM Cortex-A8 processor is a high-performance, low-power, cached application processor that provides full virtual memory capabilities. This processor implements the ARM RISC Architecture v7-A. The processor structure is shown in Figure 5.5

The Cortex-A8 implements dynamic branch prediction with branch target address cache, global history buffer, and 8-entry return stack. It also includes a Memory Management Unit (MMU) and separate instruction and data Translation Look-aside Buffers along with Level

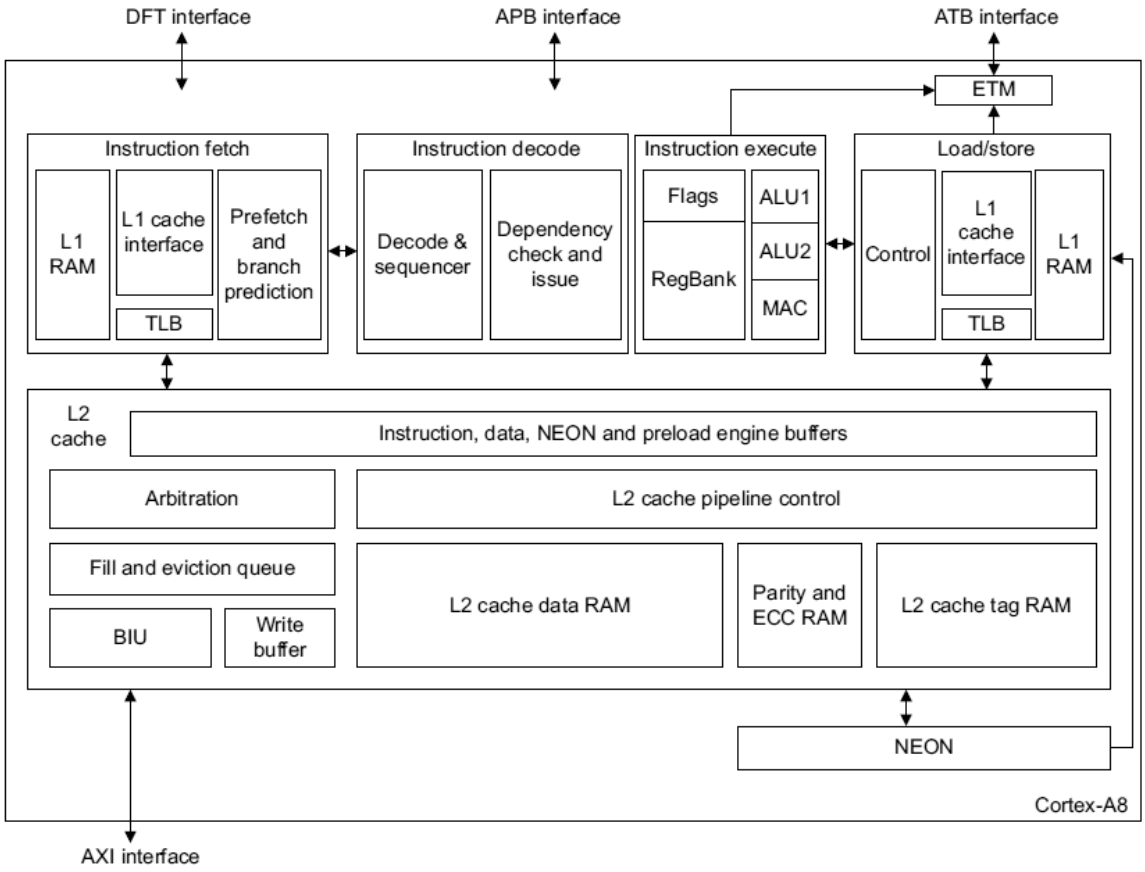

Figure 5.5: ARM Cortex-A8 Processor Structure [12].Used under fair use guidelines, 2011.

1 instruction and data caches and Level 2 cache of configurable size. The Cortex-A8 also includes a NEON pipeline for executing Advanced Single Instruction Multiple Data (SIMD) and Vector Floating Point (VFP) instruction sets.

The ARM architecture is a Reduced Instruction Set Computer (RISC) architecture. It includes typical RISC features such as large uniform register file; a load/store architecture where data-processing operations only operate on register contents not directly on memory contents; and simple addressing modes, with all load/store addresses being determined from register contents and instruction fields only.

The Cortex-A8 is a modern, complex processor. As mentioned before, one of the characteristics of non-deterministic processors is that they implement deep pipelines to improve

parallelism. The Cortex-A8 is a prime example of that, with multiple pipelines supporting a variety of coprocessors in the systems. Figure 5.6 shows the Integer pipeline.

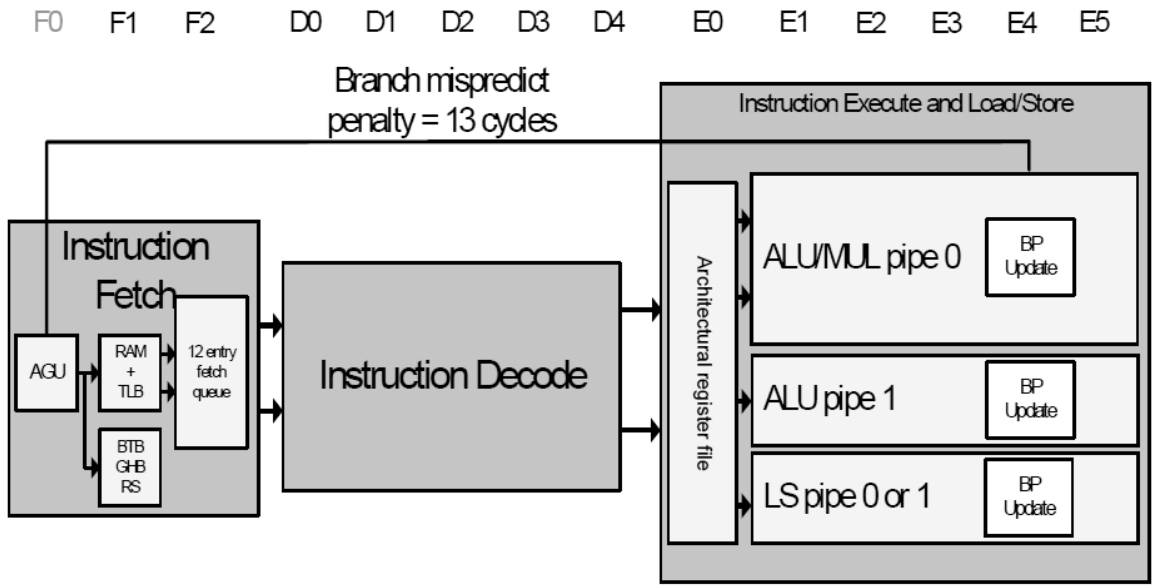

Figure 5.6: ARM Cortex-A8 13-Stage Integer Pipeline [11].
Used under fair use guidelines, 2011.

This is a 13-stage pipeline that includes instruction fetch, decoding, executing, and storing. The instruction stream is predicted in the instruction fetch unit, while instructions are ultimately executed in one of two Arithmetic Logical Units (ALUs). Such large pipeline presents a challenge to PFP as the impact of branch miss-prediction represents a 13 instruction cycle penalty. Also, the effect of random parameters in the execution is spread across multiple instruction cycles, which can add noise and uncertainty to the signatures.

From the PFP perspective, such complex device presents a significant challenge. There are several subsystems working in parallel that add noise and interference to the traces. Fortunately, modern processors are divided into different power domains, which are fed by different power rails. Different power domains allow different circuits to be controlled independently of the rest of the system, which enables significant power consumption savings and extends battery life. Having separate power domains helps PFP because it prevents

unnecessary noise and interference by strategically locating the current sensors. Figure 5.7 shows the different power domains in the OMAP3, including the ones that feed the Cortex-A8.

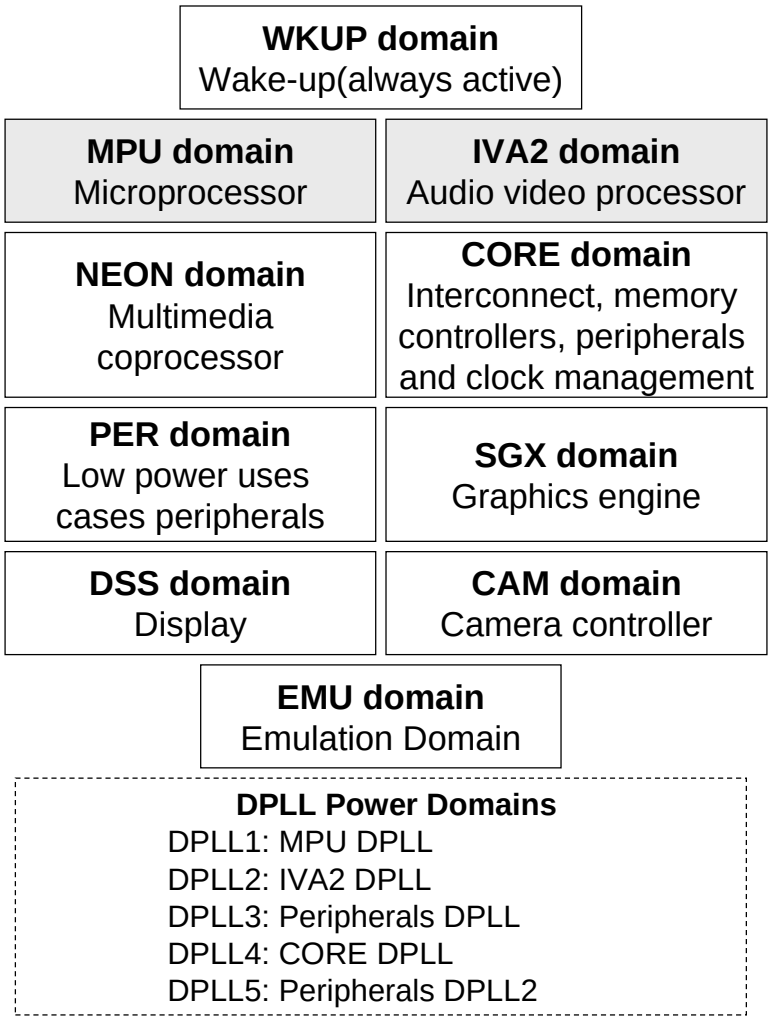

Figure 5.7: OMAP3 Power Domains. Adapted from [37]. Courtesy Texas Instruments.

It is important to mention that, due to the layout in the BeagleBoard, the MPU and IVA2 power domains, shown in gray, are fed using the same power rails. Therefore, there is significant interference from the extra modules in the captured MPU traces. This is an unavoidable problem with the current setup, but that can be easily avoided if the board itself is designed with PFP in mind and different power rails are provided for each power

domain.

Before describing the measurement setup, it is important to describe the relationship between the OMAP3 processor and the power management chip. In complex processors, it is common to have companion chips that provide specialized services. For example, the companion TPS65950 integrated power management/audio codec [38] provides power and peripheral requirements of the OMAP3 application processors. The TPS65950 provides not only power management, but also an audio codec, a universal serial bus (USB) transceiver, an ac/USB charger, LED drivers, an analog-to-digital converter (ADC), a real-time clock (RTC), and power control.

The power portion of the device contains three buck converters, multiple low-dropout (LDO) regulators, an embedded power controller (EPC) to manage the power-sequencing requirements of OMAP, and an RTC and backup module. In terms of power, the TPS65950 provides the different voltages required by the OMAP3 and the necessary power sequencing to successfully start the processor. The actual connection between the TPS65950 and the OMAP3 as it is implemented in the BeagleBoard is depicted in Figure 5.8.

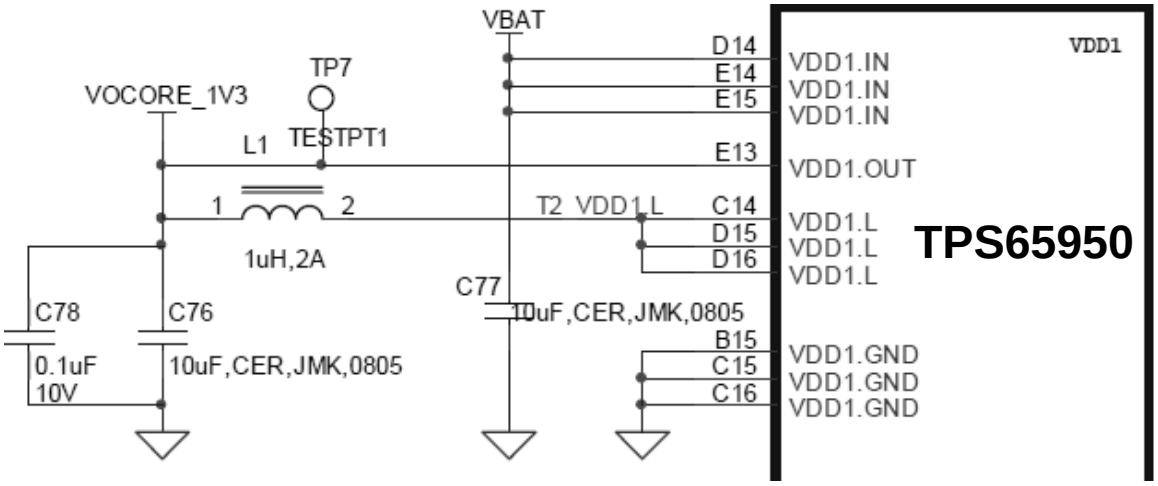

Figure 5.8: Schematic showing the main power rail from the TPS65950 power management chip to the MPU and IVA2 power domains [15].
Used under fair use guidelines, 2011.

In the BeagleBoard, the main power rail, labeled `VOCORE_1V3`, goes from the exit VDD1 of the TPS65950, after the tank circuit (RL filter) at the output of the buck converter, and feeds the MPU and IVA power domains. The `VOCORE_1V3` power rail presents an opportunity to place a current sensor that eliminates the interference from other subsystems in different power domains. The specific setup to capture PFP traces from the BeagleBoard is described in the following section.

Measurement Setup

In order to accommodate the PFP monitor, the BeagleBoard is slightly modified by cutting the main core power rail, labeled `VOCORE_1V3`, and connecting a commercial Tektronix TC-6 current probe. This setup is depicted in Figure 5.9.

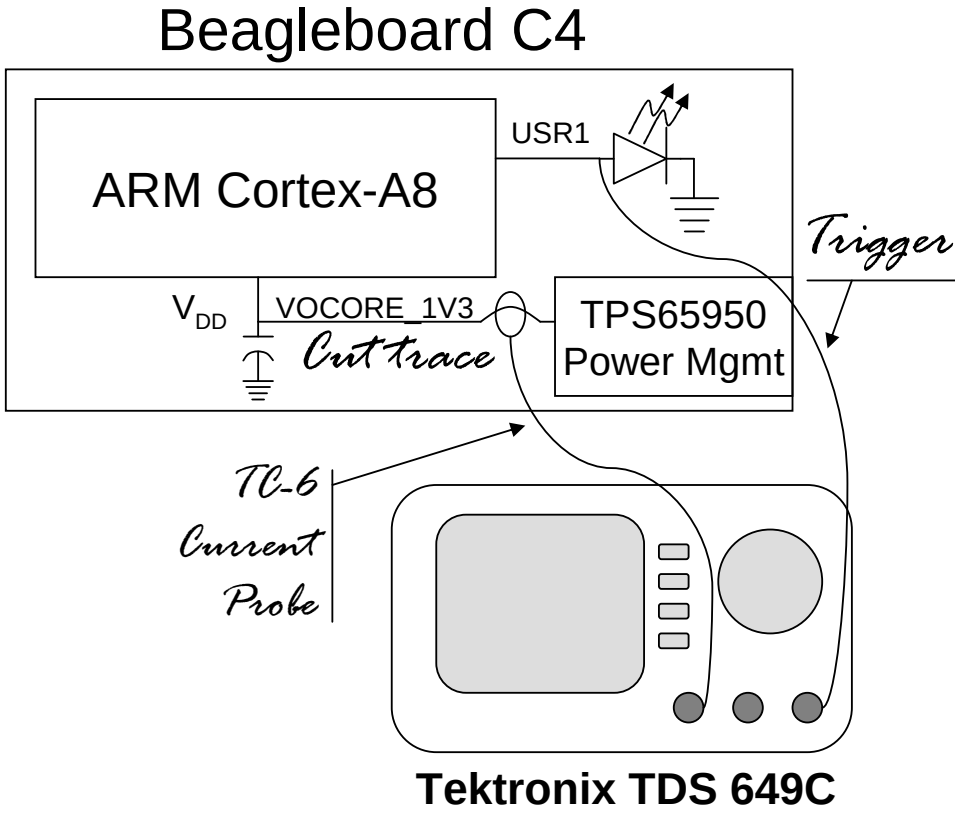

Figure 5.9: BeagleBoard PFP measurement setup

The signal captured by the current probe is then digitized using a commercial real-time oscilloscope, also from Tektronix. The oscilloscope is configured to a sampling rate of 2.5 GSPS and a total of 30K samples are collected in every trace. A physical trigger signal is provided by the BeagleBoard using one of the available LEDs (USR1). This signal is used for synchronization purposes to indicate the oscilloscope when to start capturing power traces.

5.4 Feasibility Experiment: Conditional Execution Detection in Android

The main goal of this experiment is to demonstrate the ability of PFP to monitor an Android application on a platform representative of current smart phones. The expected result for this feasibility experiment, is for PFP to detect small logic modifications made to the target application on User space.

The general approach to this experiment involves developing a basic Android app and characterizing one of its modules by extracting the its power fingerprints. The signatures are used to monitor the execution of the module at real time. An alternative version of the app, with covert functional modifications is developed and used to perform a blind test on the PFP monitor. It is important to note that the covert modification is small, with no noticeable effects on execution, timing, latency, or similar areas from the user perspective. The intention of this modification is to emulate a covert malicious presence and must be completely invisible to the user.

The covert modification is implemented by means of conditional execution, which adds potentially harmful functionality that is only executed under specific conditions. The modification is designed to emulate malicious behavior known as time or logic bombs. This type of malware is particularly dangerous as it can remain unnoticed in a system for extended periods of time, and then turned into action at a specific time, or command, and launch a coordinated

action that can have devastating consequences.

This feasibility experiment is crucial for the development of PFP technology. It provides evidence of the ability of PFP to monitor complex software structures on complex platforms. The experiment also demonstrates the feasibility of overcoming the challenges imposed by non-deterministic processing platforms in terms of modules characterization and signature extraction. Finally, this experiment will provide a reference or guideline on potential feature extraction techniques, synchronization signaling, detector design, and performance evaluation of PFP monitoring of complex software running on non-deterministic platforms.

5.4.1 Experiment Setup

The basic app developed for this feasibility experiment consists on a simple counter that displays an increasing integer on the device screen. The general structure of the app is described in Figure 5.10 and consists of a typical Android Java application with an initialization routine that prepares the screen for showing a text box and sets an integer variable used as a counter. The routine called displayCounter is in charge of incrementing the value of the counter and displaying it on the screen. This routine is configured as a recurrent task that is called every second.

The critical IncrementVal routine does not follow the traditional Java model, instead it is implemented in native C code and included as an external library by Android's NDK toolset. The IncrementVal routine, shown on the right side of Figure 5.10, is the target module for characterization and monitoring in this experiment.

As depicted in Figure 5.10, before the critical section is executed, a physical trigger signal is used to instruct the PFP monitor to start collecting power traces. Because the target application is executed in User space, the physical signal is triggered by means of the interface file common in Linux device drivers, as shown in Figure 5.11. The result of this indirect access to the hardware resources is increased timing uncertainty, as the the time between

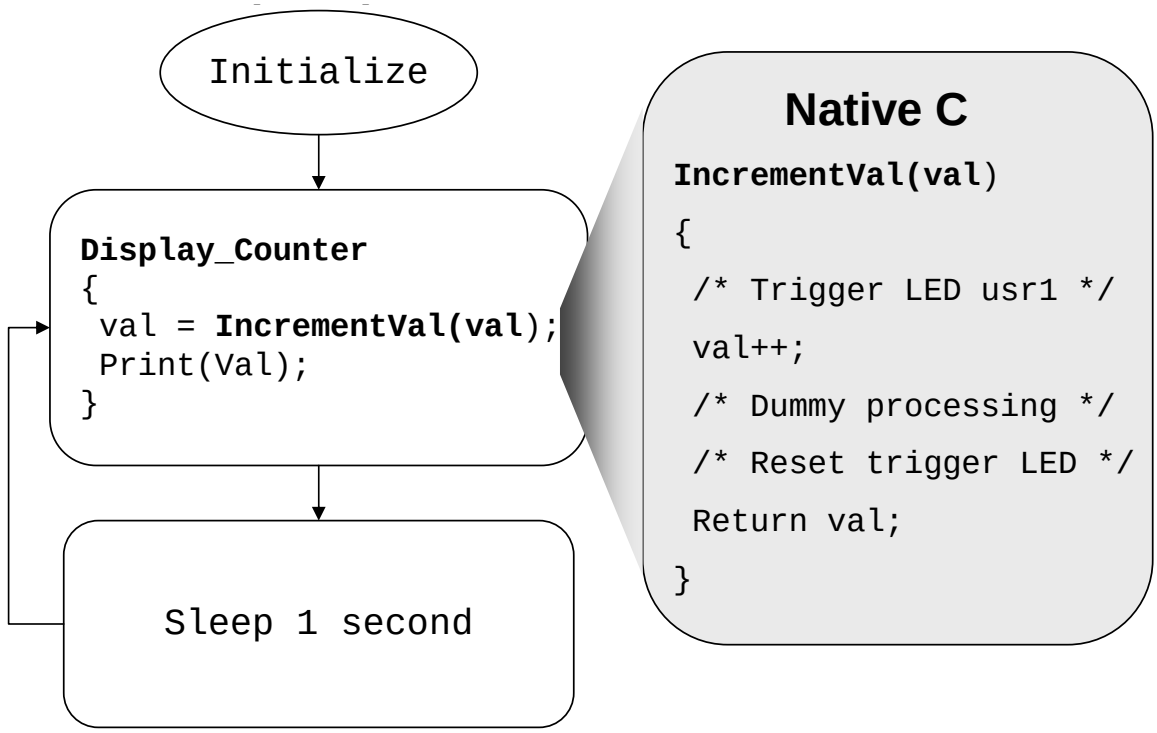

Figure 5.10: Android PFP Counter App Structure

the moment the trigger is requested and the moment the actual signal is triggered depends on the scheduler state.

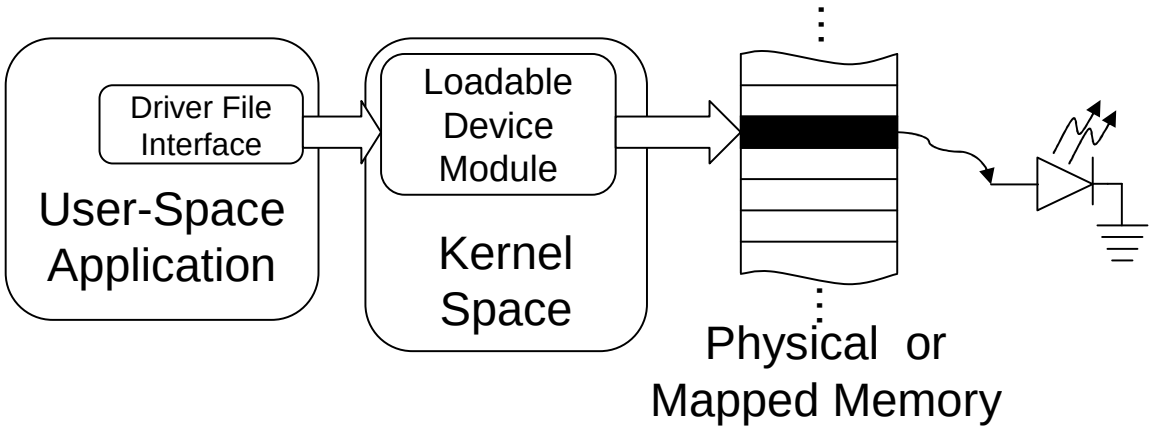

Figure 5.11: Indirect memory access using a file interface device driver to create a physical trigger signal

Introducing a Conditional Execution

In order to test the ability of PFP to detect execution deviations from trusted code, we implement a slightly tampered version of the app. The tampered app, shown in Figure 5.12, is designed to emulate a covert attack in which the intrusion remains inactive until a specific condition is met. The intrusion consists of a very simple modification in which a file is written only when the value of a counter reaches a specific value (the condition).

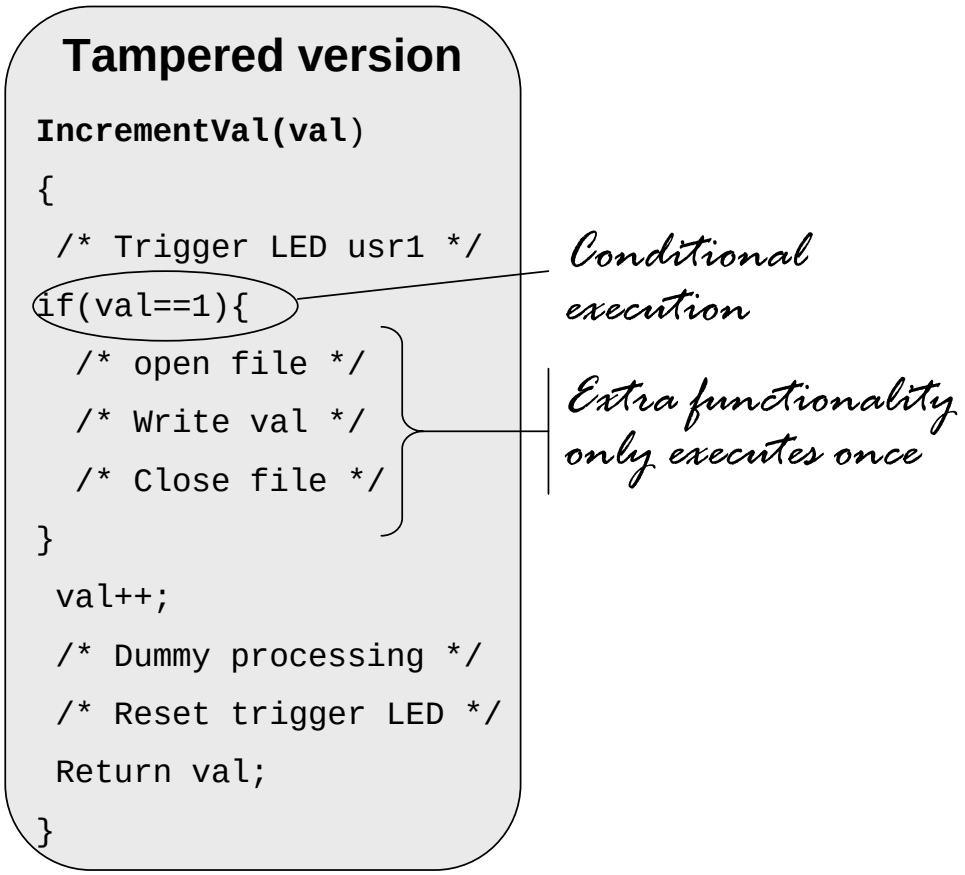

Figure 5.12: Tampered critical section in the Android PFP Counter App. The extra functionality only executes when the condition val = 1 is true

It is important to note that the additional file writing only occurs once during execution (i.e. when the counter is 1). The rest of the instances the routine is called, the condition is not met and no extra file is written. Hence, for the majority of the times the routine is called,

the only modification from a logic standpoint is an extra evaluation of a specific condition.

5.4.2 Feature and Trusted Signature Extraction

For this experiment, the discriminatory features are extracted in the frequency domain. Consider a captured sequence $r[n]$ of power measurements. The estimate of the power spectral density of the ith trace is given by $\Phi_i[k]$ which provides the power in a signal as a function of frequency. There are different options to estimate $\Phi[k]$ as shown in [53]. The reference signature, $S[k]$, is extracted from the frequency domain by simply averaging the PSD, $\Phi_i[k]$, from several traces from the execution of trusted code.

$$S[k] = \frac{1}{l}\sum_{i=0}^{l-1}\Phi_i[k] \tag{5.1}$$

The PSD of $l = 200$ traces is averaged together to yield the signature. Figure 5.13 shows the average PSD from the signature and the execution of two versions of the code.

The discriminatory features are extracted by calculating the difference between the signature and the PSD of the captured traces.

$$D_i[k] = S[k] - \Phi_i[k] \tag{5.2}$$

The average difference between the execution of the original and tampered apps is shown in Figure 5.14

The discriminatory features, X_i, ultimately being passed to the detector are the result of a moving average using a window of length m over the first f_o Hz in the PSD difference between the captured traces and the signature, as described in 5.3

$$X = \max_j \left\{ \frac{1}{m} \sum_{k=j}^{j+m-1} D[k] \right\} \quad 0 < j < f_o \tag{5.3}$$

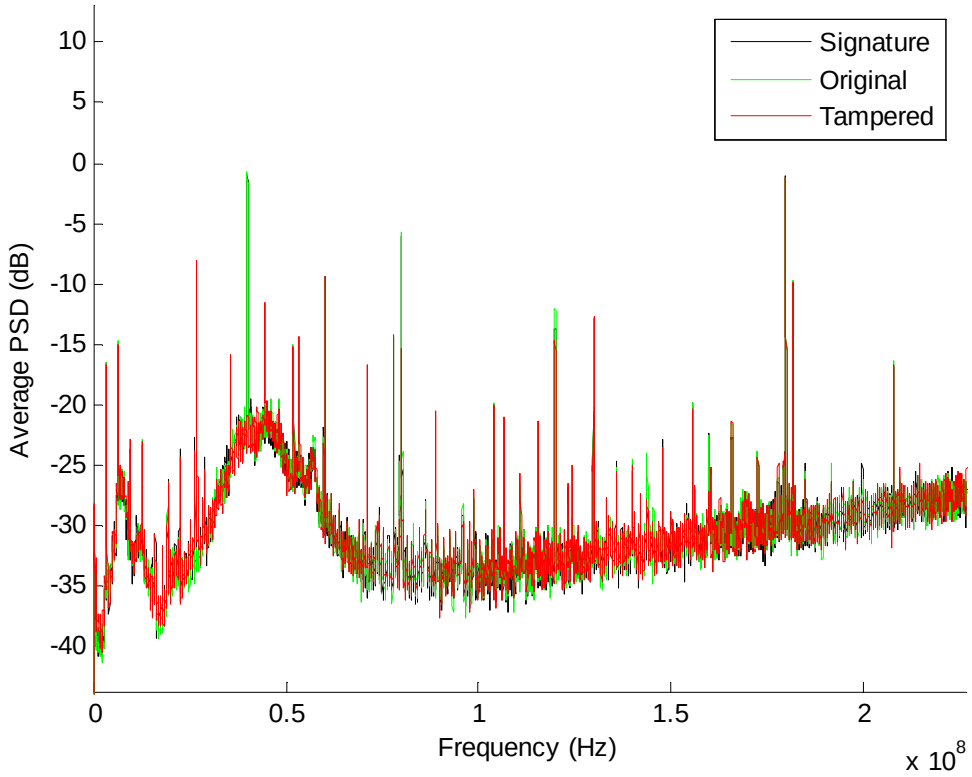

Figure 5.13: Average PSD for captured power traces from the Critical section in PFP Counter App

The PSD of the latest three test traces are averaged together before calculating the MSE. Only the first $f_o = 200MHz$ of the PSD are used in the MSE calculation. This process for signature extraction yields a mono-dimensional discriminatory feature. The discriminatory feature is passed to a detector which has been designed to determine whether the observed features fall within the expected variance or they indicate execution tampering.

5.4.3 Detector Design

Detector design is performed using the Neyman-Pearson criterion with a target probability of false alarm, P_{FA}, of 5%. The statistical analysis of the trace is performed using a sample of 200 traces from the execution of the trusted code.

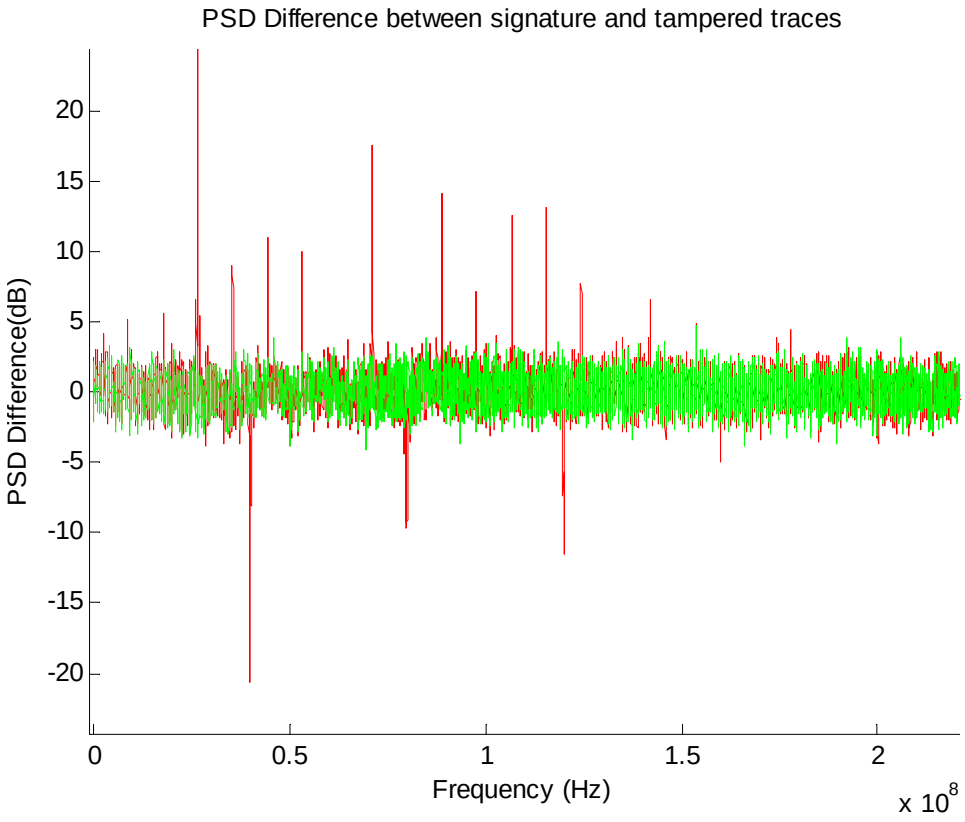

Figure 5.14: Average PSD difference from tampered and untampered traces taken from the signature extracted from the critical section in PFP Counter App

The observed distribution of training traces is fitted to a Log-normal distribution, as shown in Figure 5.15. Using the parameters estimated during the fitting process ($\mu = 1.206$ and $\sigma = 0.142994$ for the corresponding Normal distribution), the inverse probability distribution is calculated to find the threshold that yields the target 5% P_{FA}. In this case the threshold value is $thresh = 4.2258$

5.4.4 Feasibility Results

The results from monitoring the execution of the original untampered version of the Android app are shown in Figure 5.16. We can see that from all the traces captured during the test, we had only a few instances that went past the threshold to be classified as anomalies, which

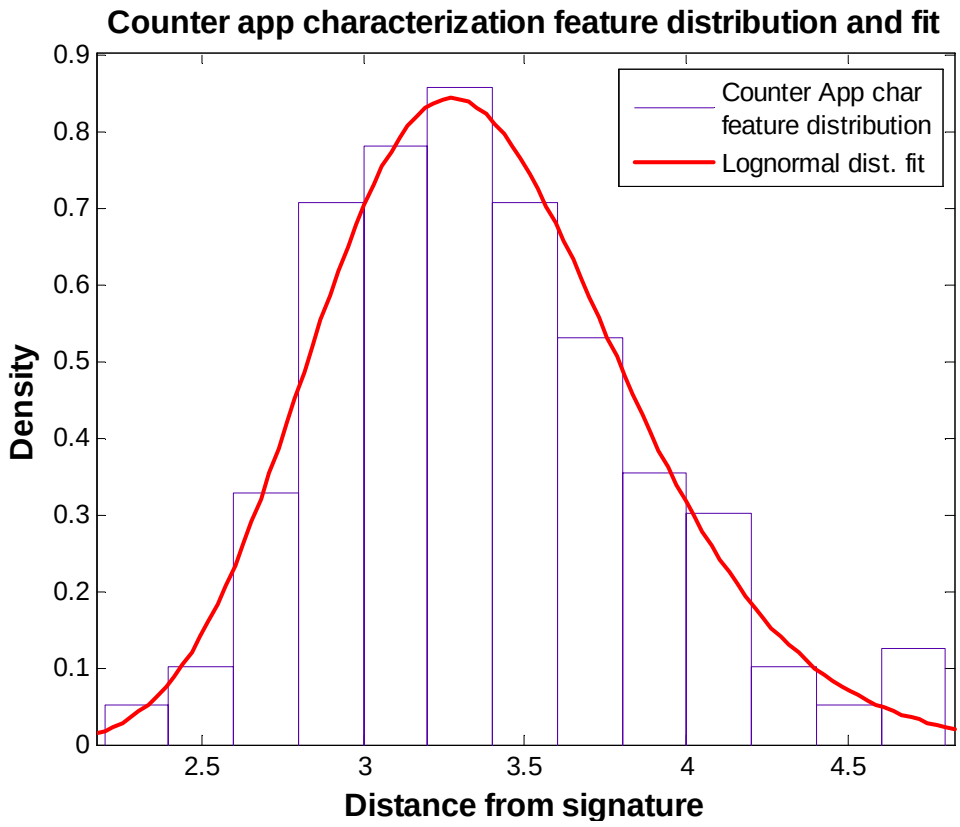

Figure 5.15: Distribution fitting during detector design for the critical section in the PFP Counter App

is consistent with the designed probability of false alarm.

The results from monitoring the tampered version of the app are shown in Figure 5.17. No instance is misclassified as authorized execution and every single execution of the tampered app is flagged as an intrusion. It is important to keep in mind that the extra code (opening and writing the file) only happened in the execution instances when the condition was met. In this case, the condition is only met the first time the routine is invoked (when val = 1). The rest of the instances, the condition was checked, and when it was not met, normal execution resumed.

In order to have a reference of the distance from the original and tampered executions, Figure 5.18 shows the monitoring results of both executions combined. The figure shows

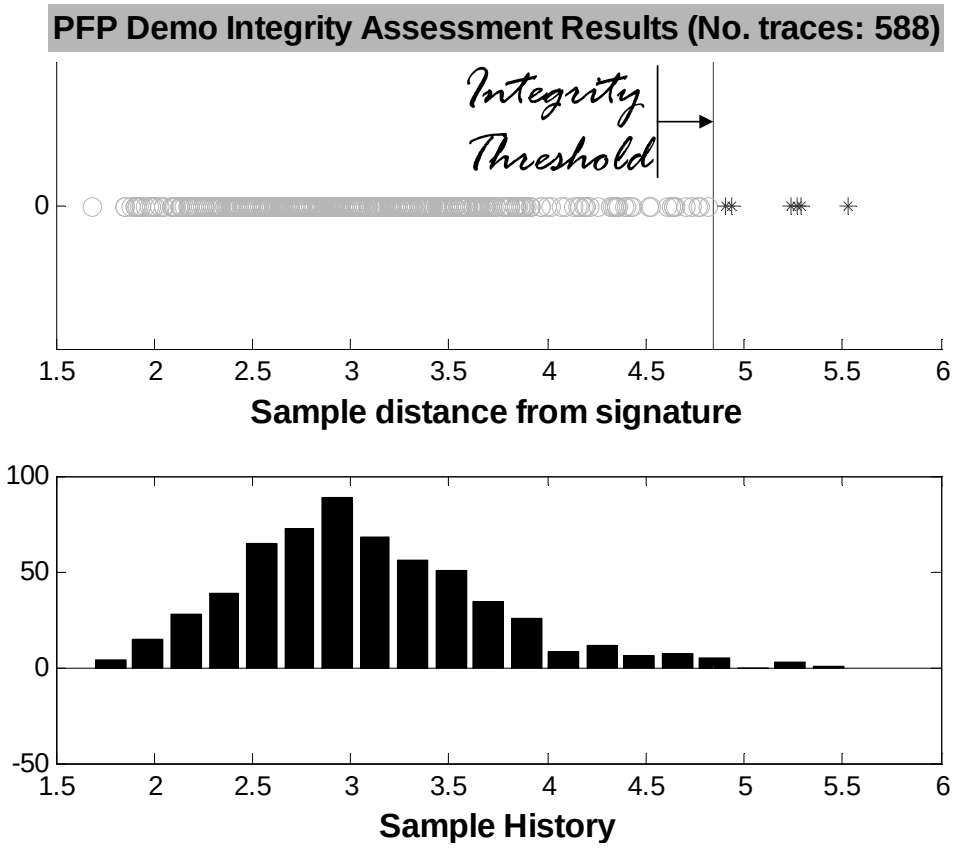

Figure 5.16: Feasibility results untampered execution PFP Counter App

a clear separation between the resulting distributions from both executions. The distance is large enough that leaves plenty of room to improve the performance of the system by changing the detector design to reduce the P_{FA}, effectively pushing the integrity threshold further away from the signature.

The reason why such a small execution variation can be detected by the PFP monitor is because the insertion of a few extra instructions affect the location in memory of the ones following, changing the Hamming distance between consecutive states compared to the original code. In other words, when the intruder checks a specific condition, it affects the flow of the program (Observer's effect) in enough fashion that the PFP monitor can determine changes in the power signatures.

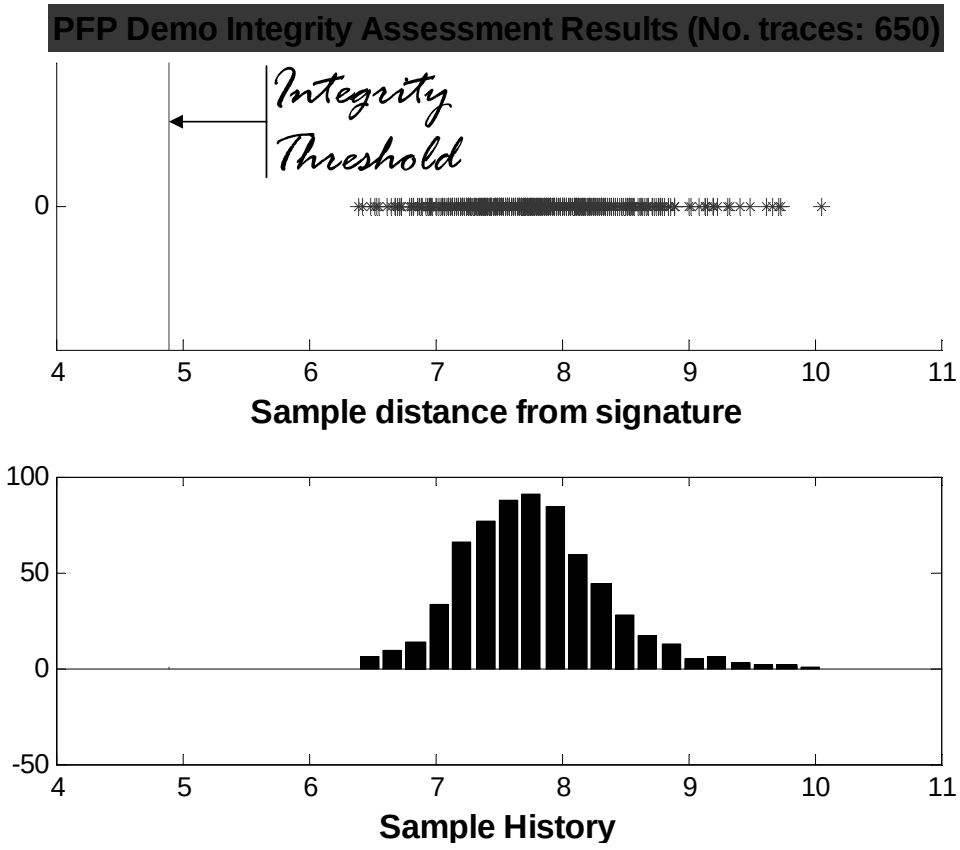

Figure 5.17: Feasibility results tampered execution PFP Counter App

These seemingly simple results are extremely important, as they demonstrate that it is possible to monitor complex platforms using FPF. They also shown that even a small intrusion, such as simply checking the value of a variable, can disrupt the signatures enough to trigger a PFP flag. This last result indicates that it is possible to use PFP to detect hidden malicious intrusions, such as time bombs, logic bombs, and kill switches before they have a chance to be activated.

These results also open the possibility of using a predetermined input to perform PFP assessment. A predetermined input in PFP monitoring (the exact same one used during characterization) reduces uncertainty as the state sequence is determined by the input. Using a predetermined input, however, also allows a potential attacker to "learn" it and hide when the PFP input is detected. Fortunately, for the attacker to detect the presence of the special

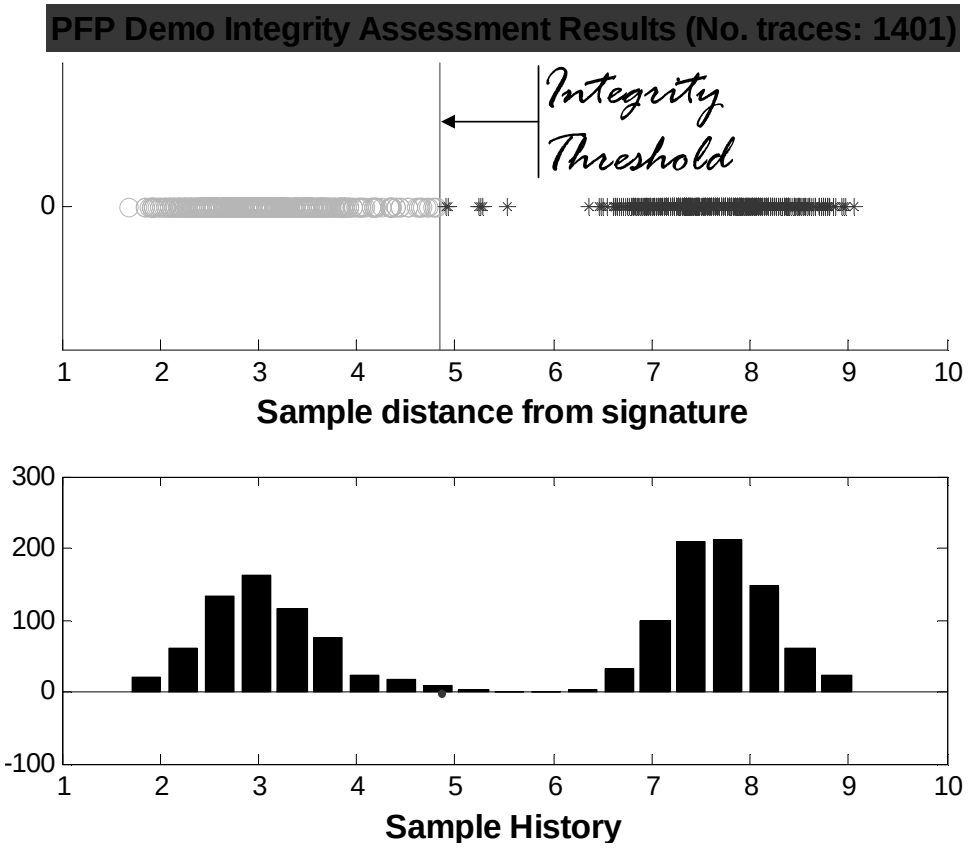

Figure 5.18: Combined feasibility results PFP Counter App

input, it has to observe it, and make the malicious execution conditional to the specific input. This is the exact scenario we described in this experiment. Hence, the attacker, by observing the input to determine if a PFP scan is taking place, would disrupt the PFP signatures and disclose its presence.

5.5 Feasibility Experiment: Real-World Malware Detection in Android

The previous experiment provided initial evidence about the feasibility of applying PFP on complex, non-deterministic processors. It also introduced the necessary techniques to detect

small execution changes. While the results are general and cover a large number of scenarios, they come from a custom app with no practical functionality. In order to provide definitive evidence of the feasibility and relevance of PFP, it is necessary to detect real malicious software attacks. That is the goal of this section.

For this experiment, the main goal is to demonstrate the ability of PFP to detect a real malicious attack under realistic conditions in an Android platform. The experimental procedure includes characterizing a core Android module during its normal execution, executing a known privilege escalation attack that exploits a vulnerability in such module, and use PFP to detect the attack in a blind test.

This section starts by describing the logical operation of the malicious attack. It also describes the characterization process for the target Android module. The performance results of the blind test are presented at the end. These feasibility experiments present the strongest evidence on the ability of PFP to monitor the integrity of sophisticated commercial devices and detect malicious attacks in a timely manner.

5.5.1 Malware Description

The malware used in this experiment is called "RageAgainstTheCage" (RATC). This is a privilege escalation attack that exploits an unchecked `setuid()` system call in the Android Debug Bridge (ADB) module. RATC gives the attacker root privileges in the compromised device, effectively granting complete control of the device. Privilege escalation attacks are also called jailbreaking attacks, as their goal is to violate the security boundaries set in place by the system design.

RATC achieved world-wide attention as one of the exploits used in the famous "DroidDream" malware [56]. DroidDream was contained in a number of malicious apps distributed in the Official Android Market. Figure 5.19 shows a news screen capture from the New York Times about DroidDream.

Figure 5.19: News about malicious apps in the official Android Market. The malicious apps contained DroidDream, which uses RATC [56].
Used under fair use guidelines, 2011.

RATC has also been used in a family of related Android malware called 'DroidKungFu'. Different from DroidDream, the different version of DroidKungFu have only been distributed in alternative Android Markets. Popular mobile antivirus solutions for Android are effective at catching DroidDream. New versions of DroidKungFu have emerged consistently, showing how easy it is for attackers to evade traditional signature-based malware detection systems. In the different versions of DroidKungFu, a simple change in the malware, such as encrypting the payload, changing command and control servers, or simply changing the malicious component's name, are enough to evade the detection systems [39].

The goal of RATC is to jailbreak out of the sandbox by acquiring Root, or super-user, privileges. The operation of RATC is very simple. It exploits a logic error in the implementation of ADB in Android version 2.2x and below. ADB is a debugging interface that allows limited access to certain OS resources, such as shell terminal to execute commands.

RATC exploits a flaw in the software's logic, making it more reliable than other exploits that rely on boundary violations such as buffer overflows. The operation of RATC is depicted in Figure 5.20. Under its normal operation, the init process launches the ADB Daemon (ADBD) when USB debugging is enabled in the OS. When ADBD is launched, it needs to perform certain tasks that require Root privileges. As soon as these tasks are completed, ADBD drops its root privileges by invoking the `setuid()` [22] system call, which effectively changes the effective ID of a process within the OS. After invoking `setuid()`, adbd changes its ID from ROOT to SHELL, with the latter having limited privileges. When a user requests a shell using ADBD, the effective ID of the shell, and all derived processes, is SHELL.

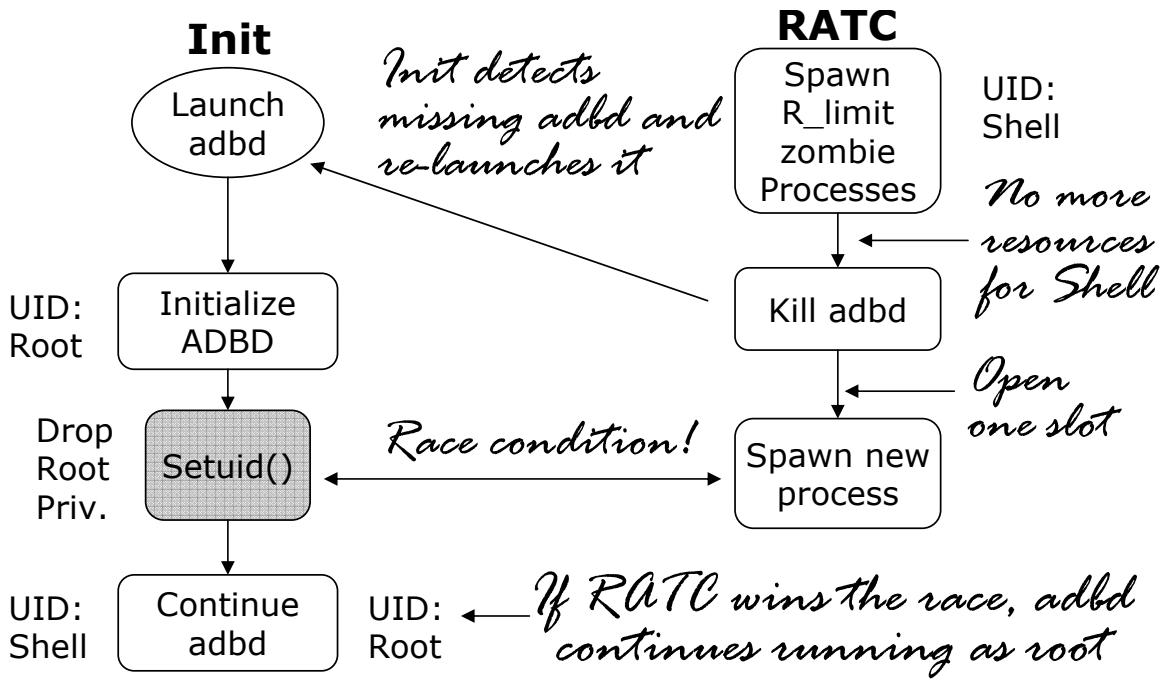

Figure 5.20: Description of RATC Logical Operation

Unfortunately, in Android versions below 3.0, the result of the `setuid()` call is not verified for correct completion. ADBS's implementation simply assumes that `setuid()` call will complete without error and drop Root privileges. RATC challenges this assumption and creates an execution condition under which `setuid()` is not able to complete. RATC is initially launched using a shell provided by ADBD with an ID of SHELL. When RATC is

launched, it finds the process ID (PID) of ADBD and then spawns a large number of zombie processes[1], until the number of processes that can be assigned to SHELL are exhausted. Once the resources have been exhausted, RATC kills ADBD. By killing ADBD, one extra slot for a SHELL process is made available. When the OS detect the termination of ADBD, init re-launches it automatically. At the same time, RATC attempts to spawn one more zombie process under SHELL ID. Because there is only one extra available slot, a race condition is created. If RATC wins the race, i.e. its last process fills the only available slot before ADBD completes the `setuid()` call, ADBD cannot drop its Root privileges. Because the exit status of the `setuid()` call is not checked, ADBD continues running with Root privileges. The next time ADBD serves a shell request, it delivers a shell with root privileges. In the event that ADBD wins the race, it will successfully drop its root privileges. in this case, the attacker needs to start the whole process all over again. The steps an attacker needs to take to complete a RATC attack are described in Figure 5.21.

From the study of the RATC operation, it is cleat that in the event of a successful attack, the expected operation of ADBD is disrupted. In other words, RATC forces an unexpected execution path in ADBD.

5.5.2 Target Profiled Code

For this experiment, PFP will monitor the execution of ADBD and look for deviations from the expected execution. In order to understand the experiment, it is necessary to describe the sections of the code being monitored. The section of ADBD that is attacked by RATC is shown in Listing 5.1. In Line 7, the unchecked call to `setuid()` that RATC exploits is executed.

```
0 int adb_main(int is_daemon)
  {
      ...
      gid_t groups[] = { AID_ADB, AID_LOG, AID_INPUT, AID_INET, AID_GRAPHICS,
```

[1] A zombie process is a process that has been terminated but whose status has not been checked by its parent, what prevents its entry from being removed from the process table

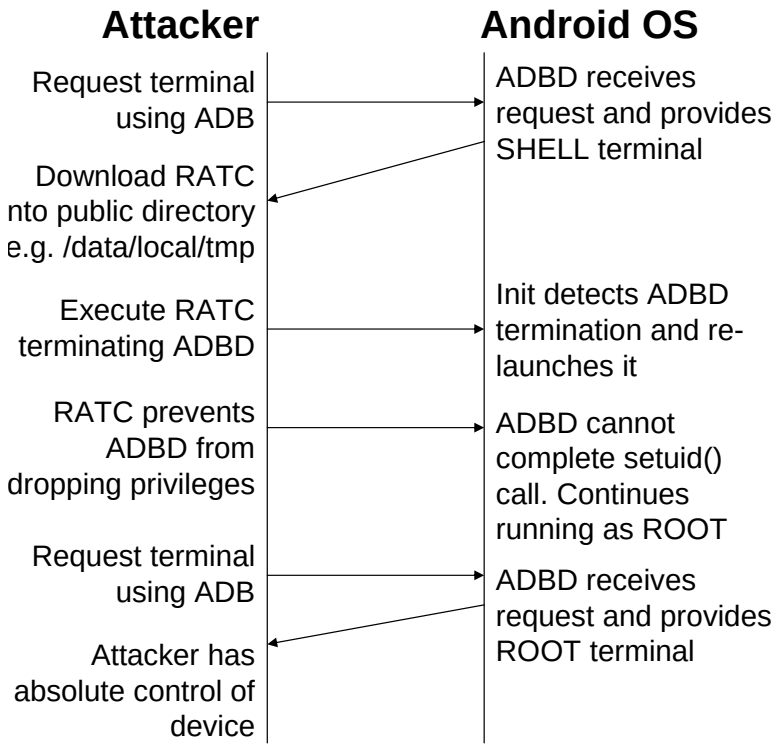

Figure 5.21: RATC Attack sequence

```
                    AID_NET_BT, AID_NET_BT_ADMIN, AID_SDCARD_RW, AID_MOUNT };
    setgroups(sizeof(groups)/sizeof(groups[0]), groups);
    setgid(AID_SHELL);
    setuid(AID_SHELL);

    /* set CAP_SYS_BOOT capability, so "adb reboot" will succeed */
    header.version = _LINUX_CAPABILITY_VERSION;
    header.pid = 0;
    cap.effective = cap.permitted = (1 << CAP_SYS_BOOT);
    cap.inheritable = 0;
    capset(&header, &cap);
    ...
```

Listing 5.1: Android ADBD Implementation Code (adb.c)

`setuid()` is a system call. Hence, it operates at the Kernel level. Inside the Kernel implementation of `setuid()`, under `kernel/sys.c` there is another call to a function named `set_user()`, which is the piece of code that actually makes the ID change. Listing 5.2 shows the `set_user()` implementation.

```
0 static int set_user(struct cred *new)
```

```
{
    struct user_struct *new_user;
    unsigned long flags; //For spin_lock
    int i;
5
    new_user = alloc_uid(current_user_ns(), new->uid);
    if (!new_user)
        return -EAGAIN;

10  if (!task_can_switch_user(new_user, current)) {
        free_uid(new_user);
        return -EINVAL;
    }
    if (atomic_read(&new_user->processes) >=
15          current->signal->rlim[RLIMIT_NPROC].rlim_cur &&
            new_user != INIT_USER) {
               free_uid(new_user);
               return -EAGAIN;
    }
20  free_uid(new->user);
    new->user = new_user;
}
```

Listing 5.2: Linux Kernel set_user implementation (kernel/sys.c)

On Line 14, there is a condition that verifies that the target ID (in the RATC attack it is SHELL) still has available resources to support one more process. If the current number of processes assigned to the target ID is equal or bigger than `RLIMIT_NPROC` and the target ID is not Root, the new ID cannot be set and the function returns a `-EAGAIN` (try again) error.

At this point the goal of PFP becomes very clear: Distinguish between both execution paths in the conditional code in Line 14, with the expected path being the one in which the assignment takes place.

In order to extract the reference PFP signatures, it is necessary to characterize the execution of the set_user() function at the kernel level. The required set of training traces are obtained with help from a scaffolding script that allows for a consistent and controlled execution of the target module. The operation of this script is shown in Figure 5.22.

The characterization script executes ADBD in a repetitive fashion making sure that the expected path within the `set_user()` call is executed. The execution environment should be as close as possible to what would be considered normal operation. The operation of the

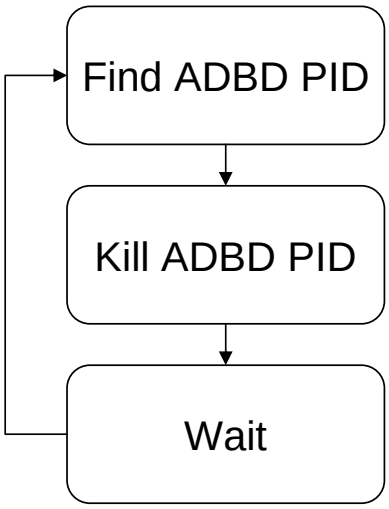

Figure 5.22: ADBD base characterization script operation

script is very similar to what RATC does, except for the resource exhaustion part. In the ADBD characterization script, the PID of ADBD is identified and used to kill the ADBD process. The script then waits for a number of seconds to give time to init to re-launch the daemon again. After the wait, the script repeats the operation. The PFP monitor captures training traces every time ADBD is initialized. These traces are later used to create the reference signature.

Because the goal for this experiment is to distinguish between two execution paths, it is appropriate to capture traces from the alternative execution path to identify the best discriminatory features. The extra set of alternative characterization traces is captured from the execution of the `setuid()` such that it returns the -EAGAIN error. This task is performed with help from special characterization scripts. The scripts force the alternative execution path in `set_user()` resulting in an `-EAGAIN` error. Two scripts are required to force the alternative execution path and their general operation is shown in Figure 5.23.

Not surprisingly, there are noticeable similarities between the operation of these last characterization scripts and the operation of RATC. This is expected because they share the same objective of forcing `setuid()` to return the -EAGAIN error. Therefore, it is necessary for

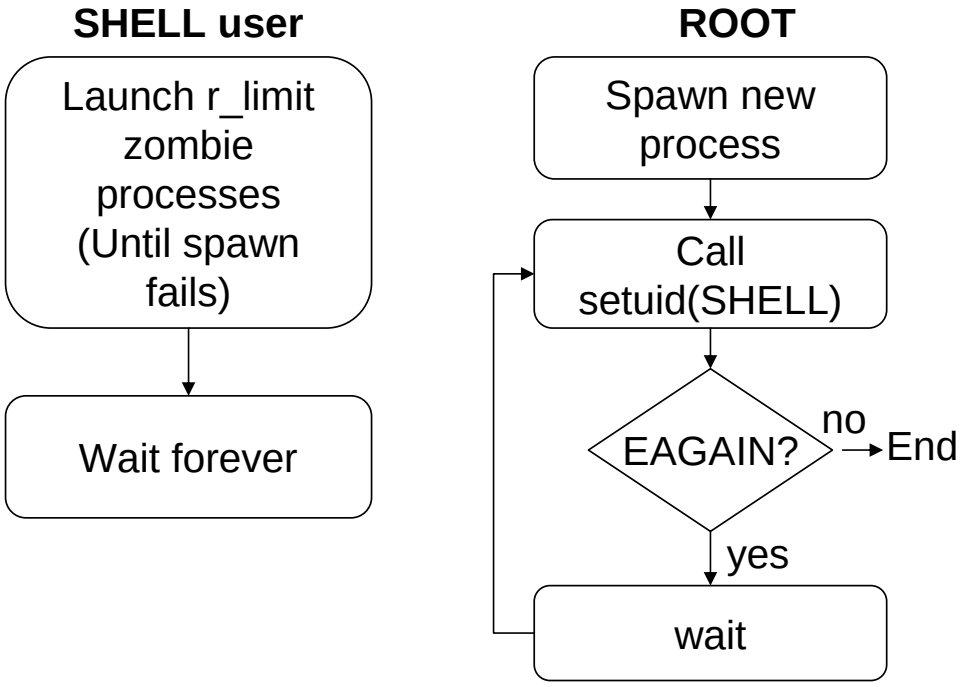

Figure 5.23: Setuid EAGAIN characterization scripts operation

the scripts to exhaust the resources of the receiving ID, just like RATC does. Once the script running as SHELL has exhausted all the resources, it goes to sleep, keeping those resources occupied. At this point, the second script, running as Root, attempts to change its ID to SHELL invoking `setuid()`. This last call fails, as there are no extra available resources for SHELL. The second script, the one calling `setuid()`, continues attempting to change its ID to SHELL until enough traces have been collected.

Signaling and Synchronization

In order to improve the determinism and facilitate the synchronization of the PFP characterization traces, the code in `adb.c` and `kernel/sys.c` was slightly modified. The modifications are highlighted in the pseudo code shown in Figure 5.24, with the target code shown in red and the PFP modifications in blue.

To improve determinism, interrupt handling was disabled before the target code is executed,

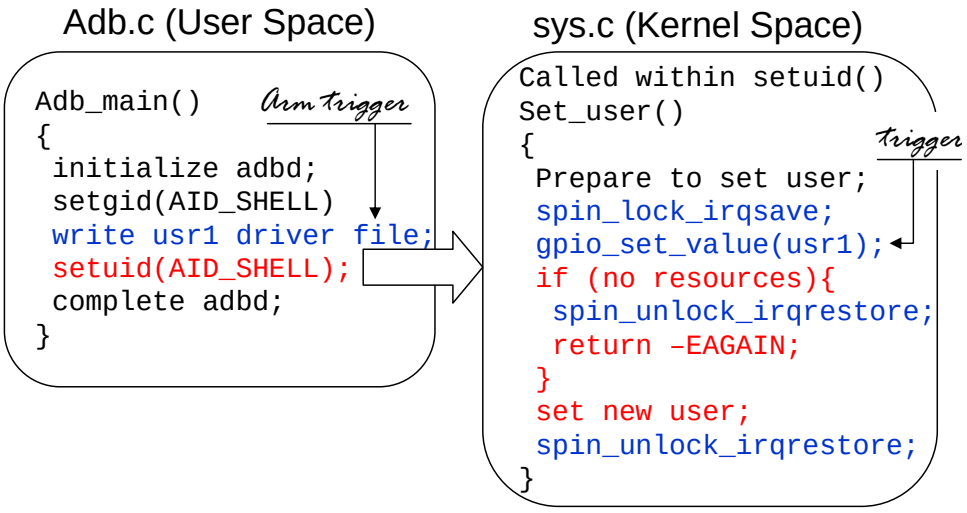

Figure 5.24: Pseudo code for the operation and modification of ADB and required system calls being monitored

thus, eliminating the possibility of the processor disrupting the execution of the target code and the power traces. The interrupts are disabled by calling `spin_lock_irqsave`, a kernel-locking utility that disables the interrupts while also saving the interrupt flags so they can be returned to its previous state. After the target code is executed, `spin_unlock_irqsave` is called to restore the interrupts to their previous state.

The modifications required to improving synchronization are a little more complex. Similar to previous experiments, a physical signal is used to trigger the monitor to start collecting samples. For this experiment, the trigger is armed in the User space before the call to `setuid()` is made. In order to interact with the USR1 LED from the User space is necessary to write the appropriate value into the device driver interface files. Once the trigger is armed, the start of the target code execution at the Kernel level can be announced to the monitor by changing the status of the usr1 LED. Because the target code and trigger are executed at the kernel level, there is no need to use the device driver interface file. At this level, the trigger is simply activated by writing directly to the GPIO port using the utility `gpio_set_value()`.

The modifications to improve determinism and provide synchronization signaling are the only

overhead, or performance penalty, introduced by PFP monitoring into the target system.

5.5.3 Feature Extraction

Similar to the previous experiment, the discriminatory features are extracted from the frequency domain by calculating the power spectral density (PSD) of the captured traces. As mentioned before, the PSD is used because it is less affected by small timing shifts that could disrupt time-domain features. Once the PSD is extracted from each trace in the training set from the expected execution, the reference signature is created by averaging together the PSDs. Figure 5.25 shows the reference signature along with the averaged PSD from both execution paths used to characterize the execution.

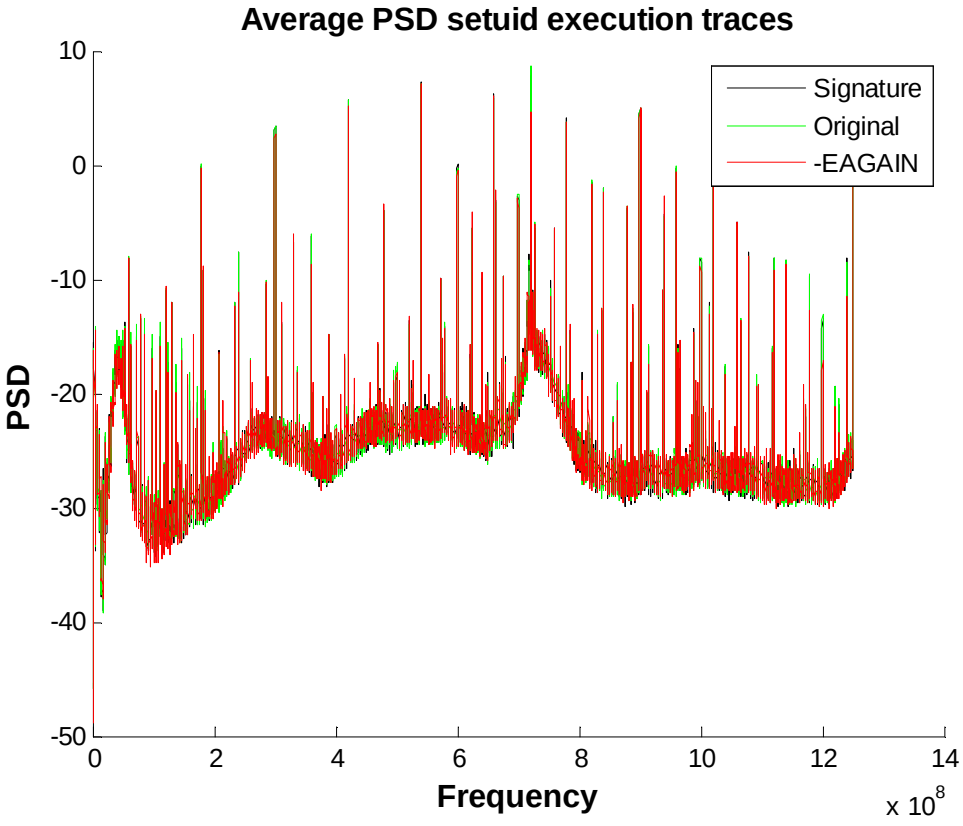

Figure 5.25: Average PSD for different execution paths in setuid

In Figure 5.25, it is difficult to see any significant separation between the averaged signatures from both execution paths. In order to better compare the resulting traces, Figure 5.26, shows the difference between the averaged PSD from the expected and -EAGAIN execution paths.

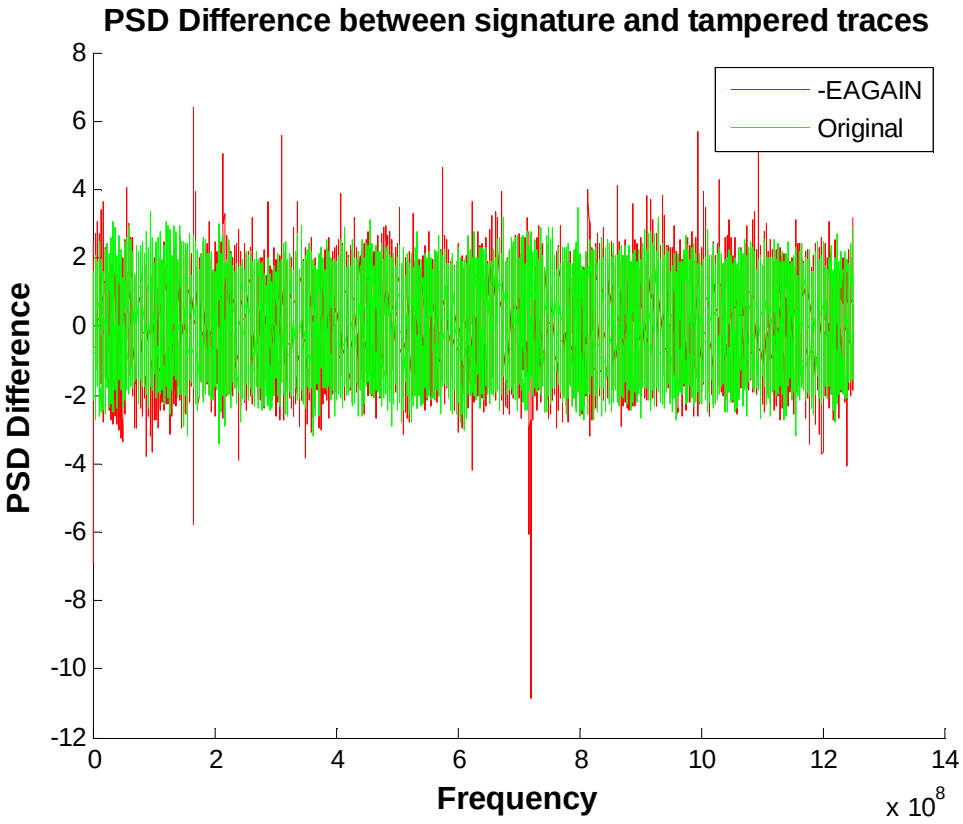

Figure 5.26: Average PSD Difference for captured power traces from different execution paths in setuid

In Figure 5.26, exactly around 720 MHz, there is a significant difference between the traces. This section of the PSDs is shown in detail in Figure 5.27. Notice that traces from the expected, or normal, execution appears as noise, while the traces from the -EAGAIN execution present a clear, consistent bias.

The PSD difference from the expected signature around 720 MHz, exactly as shown in Figure 5.27, is used as the discriminatory feature for this experiment. The same moving-

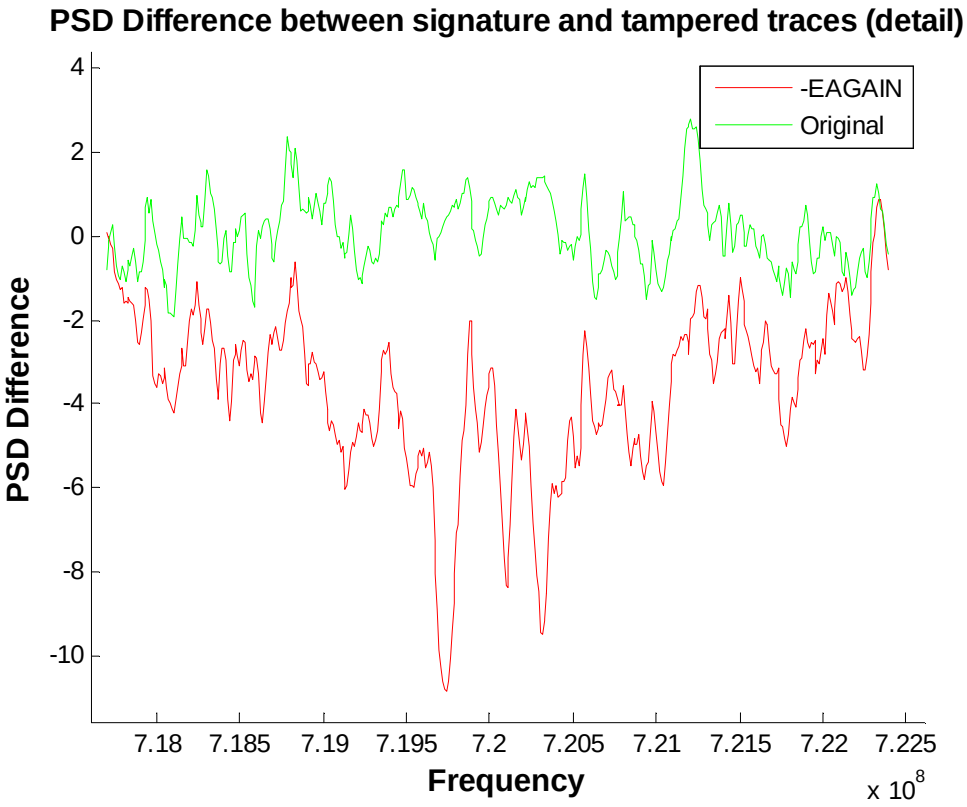

Figure 5.27: Detail from average PSD Difference for captured power traces from different execution paths in setuid

average metric described in 5.3 for the previous experiment is used to calculate the final discriminatory features, X, passed to the detector.

5.5.4 Detector Design

A similar process based on the Neymann-Pearson criterion as the one used in the previous experiment is applied to determine a threshold that yields a determined P_{FA}. For detector design, discriminatory features from a fresh set of training traces from the expected execution of `setuid()` are captured. The resulting sample distribution from the training traces is fitted to a Log-normal distribution, as shown in Figure 5.28.

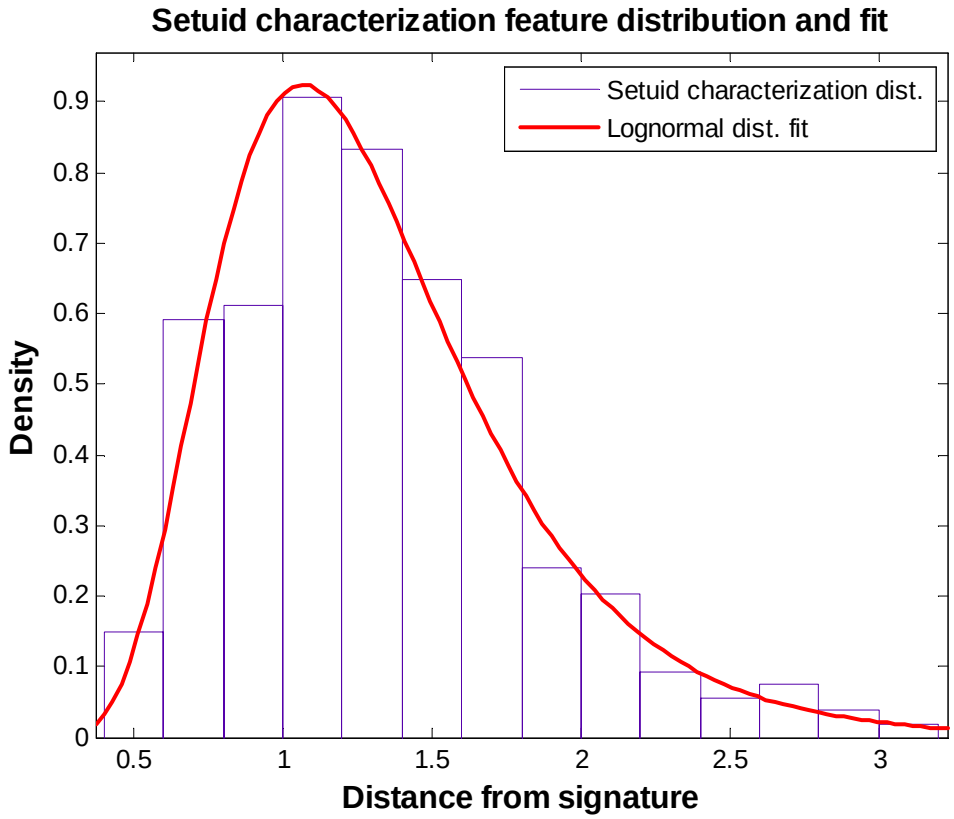

Figure 5.28: Distribution fit and detector design for expected setuid execution path

The resulting estimated parameters from fitting the Log-normal distribution are $\mu = 0.206251$

and $\sigma = 0.376927$ for the mean and variance of the corresponding Normal distribution. Assuming a target $P_{FA} = 0.05$, the threshold value is calculated using the inverse cumulative probability distribution of the Log-normal distribution with the estimated parameters. **The resulting threshold value is 2.2847**. Once the signature and integrity threshold are available, the PFP monitor is ready to assess the integrity of the ADBD execution.

5.5.5 Feasibility Results and Performance Evaluation

In this section, the performance of the PFP monitor in detecting deviations from the expected execution of ADBD is evaluated. Traces from the execution of both logic paths within `setuid`, the expected execution (original) and the tampered execution (-EAGAIN), are captured and evaluated. Figure 5.29 shows the result from 100 instances from both observed sample distributions of the discriminatory features. The integrity threshold is plotted in red for reference. The results correspond to traces captured from a single execution instance of the target code. There is no extra averaging to eliminate noise.

During the detector design process, we fixed the probability of false alarm to 5%. Using the resulting detector (threshold), the observed missed detection rate matches that 5%. In order to have a better picture of the monitoring performance using the selected discriminatory features, we fit the observed distribution from the -EAGAIN execution of the target code to a Weibull distribution. The estimated parameters for the fitted Weibull distribution are $a = 5.6706$ and $b = 4.42233$. The observed distributions with their respective fitted distributions are shown in Figure 5.30.

In order to better evaluate the sensitivity of the system, the receiver operating characteristic (ROC) curve is provided in Figure 5.31. In this figure, the resulting P_{FA} and P_D from 100 observations are shown in a dotted line, while the corresponding sensitivity using the estimated distributions is shown with a solid line. In general, the estimated performance of this PFP monitor in terms of the probability of correctly detecting an execution anomaly is over 95%.

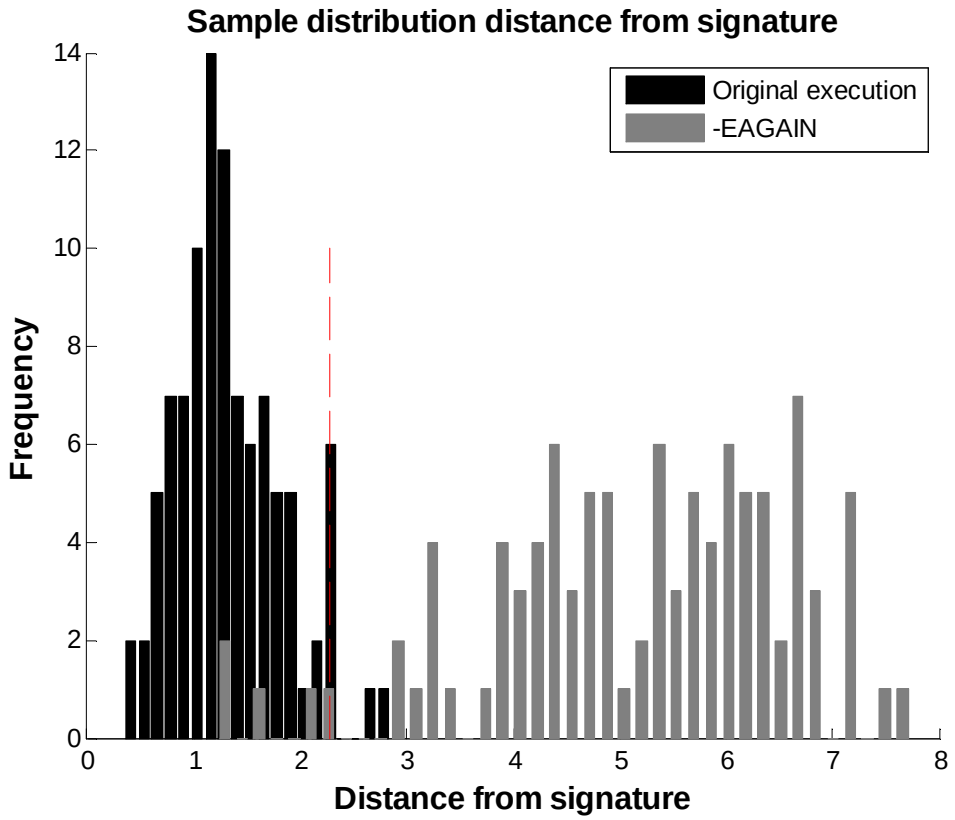

Figure 5.29: Combined feasibility results setuid PFP execution monitoring

These results provide patent evidence of the ability of PFP to detect a successful RATC privilege escalation attack. Detection happens as soon as the attacker manages to break out of the application sandbox imposed by Android. PFP is so effective at detecting this attack that RATC was detected even before the attacker could verify of the result of the race condition. The integrity violations was also detected before the attacker can use the newly acquired root privileges for malicious purposes.

It is important to keep in mind that these results are exclusive to the specific discriminatory features selected. Furthermore, it is possible to shift the integrity threshold to either side depending on the tolerance of the system to the different errors, false alarm or missed detection.

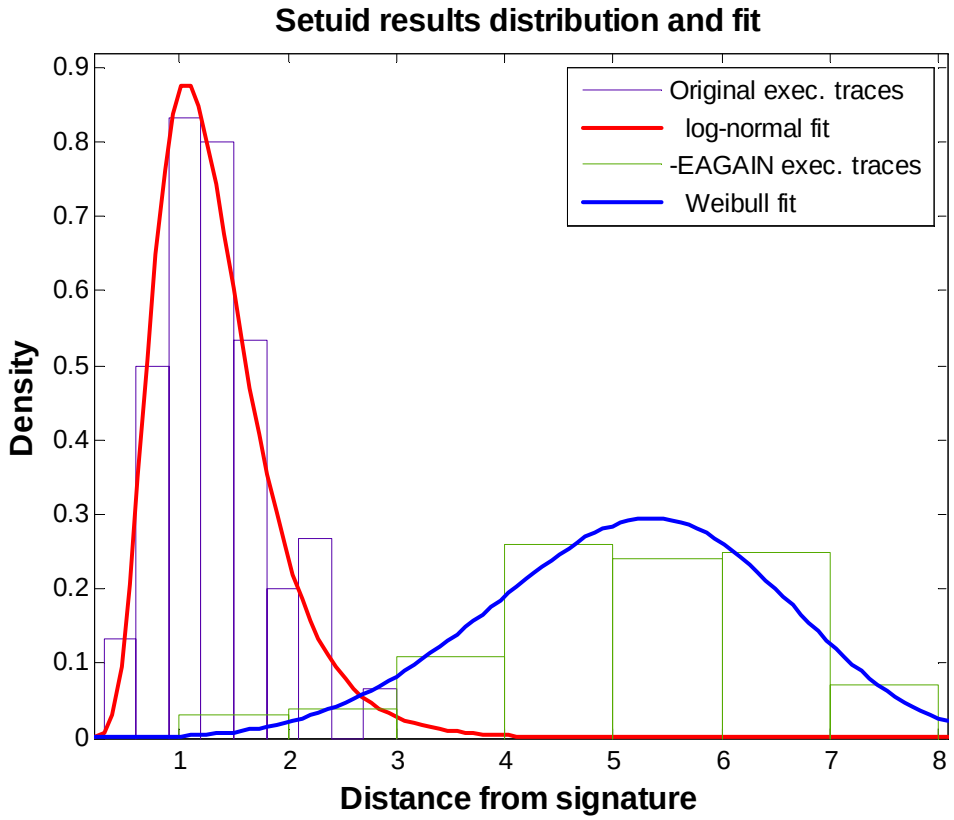

Figure 5.30: Performance evaluation setuid PFP execution monitoring

5.6 Conclusions

This chapter described the general operation of PFP on non-deterministic computing platforms. Non-deterministic platforms are processors with complex architectures capable to dynamically adapt their operation in order to improve performance. This dynamic behavior adds uncertainty from the PFP perspective. Due to the extra execution uncertainty, PFP monitors need to be designed with feature extraction techniques that are robust against time-domain variations.

This chapter presented the results from two feasibility experiments on a BeagleBoard running the Android operating system. The BeagleBoard is a popular open-source research platform that includes an OMAP3 processor, the same processor used in several commercial

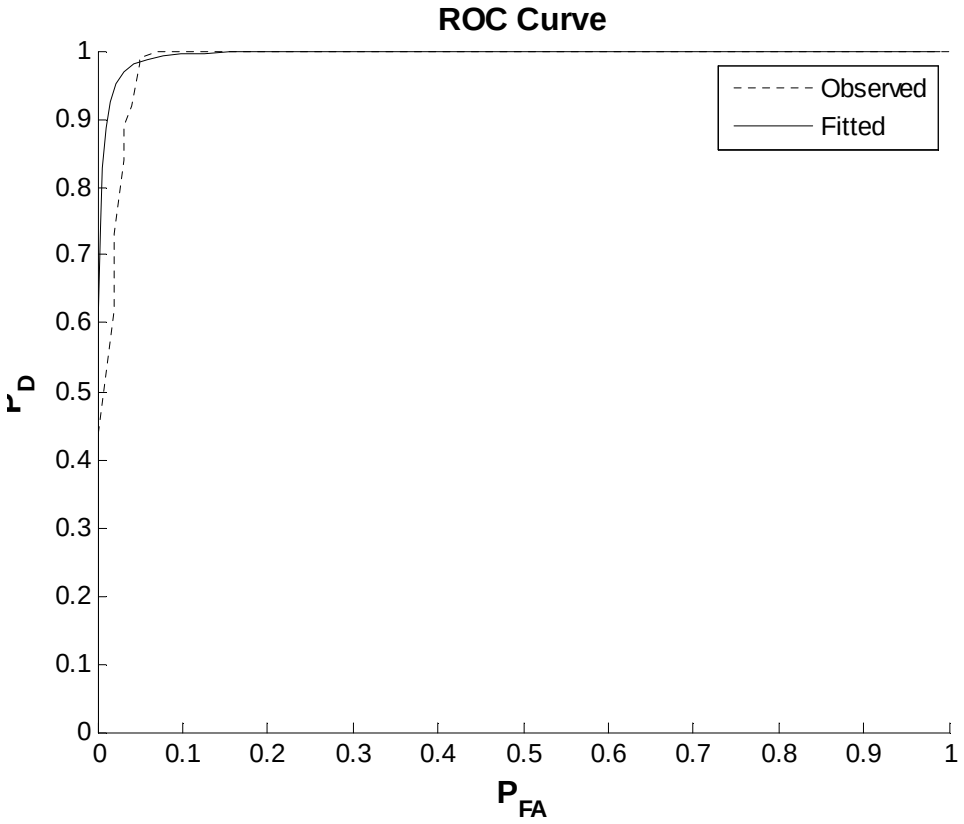

Figure 5.31: ROC curve setuid PFP execution monitoring

products. The first experiment aimed at detecting the execution of malicious software that is only triggered under specific conditions. The objective is to detect the execution deviation even in the case when the necessary conditions are not met and the extra malicious functionality is not executed. In the experiment, a basic custom Android app was developed and characterized for PFP monitoring. A second, tampered version of the app is developed. In this tampered version, extra functionality is added (malicious intrusion) conditioned to specific conditions. The PFP monitor was successful at detecting the intrusion even when the condition was not met, resulting on the extra code not being executed. This result not only demonstrates the ability of PFP to detect conditional execution, but also opens the possibility of using a predetermined input to perform integrity monitoring.

An attacker that attempts to hide malicious activity in the presence of the known PFP

input will affect the power consumption due to the Observer's effect enough to trigger the PFP monitor. Therefore, a PFP monitor is still be effective at detecting intrusions with conditional actions, such as time and logic bombs.

For the second experiment, the goal is to detect real Android malware used in the wild. The selected malware is called RageAgainstTheCage, which is a privilege escalation attack that exploits an unchecked `setuid` call in the Android Debug Bridge. This attack is used by the DroidDream malware, which was found in some malicious repackaged apps distributed from the official Android Market. In order to assess the integrity of the system, it is necessary to characterize and monitor the execution of the setuid system call at the kernel level. The PFP monitor is able to reliably detect a successful jaibreaking attack with over 90% accuracy with a single execution instance, while maintaining a probability of false alarm below 5%. The results not only demonstrated the ability of PFP to detect real malware in a practica scenario, but also showed that by characterizing and monitoring different execution paths within the Kernel, it is possible to extract very accurate behavioral signatures[2] from all execution levels.

[2]Behavioral signatures are descriptions of the different sequence of coarse functions being executed

Chapter 6

Conclusions

Cyber security is a critical element in national security and reliable integrity assessment is a fundamental requirement for an effective strategy. In this dissertation we introduced, developed, and demonstrated a novel technique for integrity assessment called Power Fingerprinting (PFP). PFP monitors side-channel information from the processor's dynamic power consumption and performs anomaly detection using pre-characterized signatures to determine whether the integrity of the system has been compromised. PFP is implemented by an external device, not only reducing the overhead on the target processor and making it applicable to platforms with severe resource constraints, but also isolating the monitor from attacks.

PFP provides significant visibility into the execution status of the processor, making it very difficult for an attacker to evade the monitor. Furthermore, because of its reliance on anomaly detection from trusted references, FPF does not need specific information about the specific malware or attack vector. Throughout this document, the main operational principles and methods for PFP are described in detail and demonstrated, which represent the main contributions of this work.

6.1 Main Contributions

The general theory of PFP is presented in Chapter 3, along with the general techniques and analysis that support and enable PFP. In this chapter the general PFP monitoring architecture is developed, along with the mechanisms to characterize software execution, extract unique signatures, and design optimal detectors an enable an external monitor to asses the integrity of a target device. Other necessary techniques to implement PFP were also described in Chapter 3, including tracking and compensating environmental changes and signaling for synchronization and identification.

The techniques and approaches introduced in Chapter 3 are put to practice and demonstrated in the next chapters. In Chapter 4, the feasibility of PFP on deterministic platforms was demonstrated by implementing a reference monitor using bench-top test equipment. In these experiments, the target is a commercial radio platform with a basic PIC processor. The PFP monitor is able to detect, with a negligible probability of error with that specific configuration, operational and configuration changes that impact security setting and spectral emissions. Also in Chapter 4, a mechanism to improve the sensitivity of PFP by characterizing the specific way a given platform consumes power was described and used to evaluate the minimum sensitivity to a change in the execution status. The pre-characterization is performed by means of a linear transformation into a transformed space that emphasizes the sections of the traces that carry the most discriminatory information and ignores redundant information. The necessary transformation matrix can be designed using LDA or PCA.

Chapter 5 expands the techniques and architectures developed in previous chapters to demonstrate the feasibility of PFP on complex processing platforms. Complex platforms are challenging for PFP because there are more architectural elements and subsystems that add interference and uncertainty to the traces. In spite of these challenges, a PFP monitor was able to asses the execution integrity of an Android application and Linux Kernel modules running on a complex, high-performance processor. This development provides the foundation for the culmination of this work, where real malicious software performing a privilege

escalation attack is immediately detected by the PFP monitor. The sensitivity of the monitor is significant, as it is capable of detecting the attack with over 90% chance, while limiting the probability of a false alarm to less that 5%.

6.2 Final Remarks

With its ability to assess the execution integrity of a system from outside the processor scope, PFP can add an additional level of security to critical cyber systems, especially for the currently neglected area of embedded devices. PFP provides a tool for the detection of stealth intrusions and helps preventing a scenario where a compromised critical system is still trusted to perform its mission. PFP has the potential to play a significant role in a defense-in-depth security architecture, by supporting complementary security approaches. PFP is a monitoring approach that gives system administrators a chance at detecting malicious intrusions and responding accordingly. The appropriate response to an integrity violation, however, depends completely on the nature of the system and the criticality of its operation.

As shown in this work, PFP is capable of assessing the integrity of embedded devices with high sensitivity, but it is important to consider that it requires a significant amount of work in terms of characterization. The work performed for this dissertation covers only embedded platforms and is not expected to be directly applicable to all systems. Significant research is still necesary to make PFP a practical deployable technology. For example, it is necessary to evaluate more feature extraction techniques to identify the best approaches for different platforms and modules. It is also necessary to crate a signature library of critical modules in common platforms and also to devise techniques to safely update the PFP signatures on deployed devices along with schedule software updates. Furthermore, there will be issues to overcome as the processors and operating systems become more complex. It is not unreasonable, however, to expect to overcome these issues with improved sensing and synchronization mechanisms as well as more complex signal detection and classification

techniques.

This dissertation presents only a starting point in developing PFP. There is plenty of work left in developing a general theory of PFP. We expect more researchers to join the study and development of PFP, eventually leading to an ecosystem in which semiconductor manufacturers and application developers integrate PFP into their designs. Such PFP ecosystem will facilitate the characterization and monitoring of critical embedded systems, ultimately improving their security and trustworthiness.

Bibliography

[1] Carlos R. Aguayo Gonzalez and Jeffrey H. Reed. Power fingerprinting in SDR and CR Integrity Assessment. In *IEEE Military Communications Conference (Milcom)*, 2009.

[2] Carlos R. Aguayo Gonzalez and Jeffrey H. Reed. Detecting unauthorized software execution in SDR using power fingerprinting. In *IEEE Military Communications Conference (Milcom)*, 2010.

[3] Carlos R. Aguayo Gonzalez and Jeffrey H. Reed. Power fingerprinting in unauthorized software execution detection for sdr regulatory compliance. In *Wireless Innovation Forum Technical Conference*, 2010.

[4] Carlos Aguayo Gonzlez and Jeffrey Reed. Power fingerprinting in sdr integrity assessment for security and regulatory compliance. *Analog Integrated Circuits and Signal Processing*, 69:307–327, 2011. 10.1007/s10470-011-9777-4.

[5] Mark Aiken, Manuel Fähndrich, Chris Hawblitzel, Galen Hunt, and James Larus. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness*, MSPC '06, pages 1–10, New York, NY, USA, 2006. ACM.

[6] Open Handset Alliance. Official Website. Available at: *http://www.openhandsetalliance.com/*.

[7] Ross Anderson and Markus Kuhn. Tamper resistance: a cautionary note. In *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association.

[8] Android. Developer's Website. Available at: *http://developer.android.com/index.html*.

[9] Android. Security Overview. *http://source.android.com/tech/security/index.html*.

[10] Android. Android dev guide. http://developer.android.com/guide/basics/what-is-android.html, 2008.

[11] ARM. Architecture and implementation of the arm cortex-a8 microprocessor. White Paper, 2005.

[12] ARM. Cortex-a8 technical reference manual. Revision:r2p1 (ARM DDI 0344D), 2007.

[13] ARM. Arm architecture reference manual. ARMv7-A and ARMv7-R edition (ARM DDI 0406B), 2010.

[14] Atmel. Atmel Website. *http://www2.atmel.com*.

[15] Beagleboard.org. Beagleboard system reference manual rev c4. Reference Manual, 2009.

[16] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton Kaliski, editor, *Advances in Cryptology CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0052259.

[17] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 37–51, Berlin, Heidelberg, 1997. Springer-Verlag.

[18] A. Bose, X. Hu, K.G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *ACM Mobisys'08*, 2008.

[19] T.K. Buennemeyer, M. Gora, R.C. Marchany, and J.G. Tront. Battery exhaustion attack detection with small handheld mobile computers. In *IEEE International Conference on Portable Information Devices*, pages 1 – 5, May 2007.

[20] T.K. Buennemeyer, G.A. Jacoby, W.G. Chiang, R.C. Marchany, and J.G. Tront. Battery-sensing intrusion protection system. In *Information Assurance Workshop, 2006 IEEE*, June 2006.

[21] L. Cavallaro, P. Saxena, and R. Sekar. On the limits of information flow techniques for malware analysis and containment. In *Springer, Lecture Notes in Computer Science*, volume 5137/2008, pages 143–163, 2008.

[22] H. Chen, D. Wagner, and D. Dean. Setuid demystified. In *11th USENIX Security Symposium*, page 171190, Aug. 2002.

[23] E. Chi, A.M. Salem, R.I. Bahar, and R. Weiss. Combining software and hardware monitoring for improved power and performance tuning. In *Interaction Between Compilers and Computer Architectures*, pages 57 – 64, February 2003.

[24] Jean-Franois Dhem, Franois Koeune, Philippe-Alexandre Leroux, Patrick Mestr, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications*, volume 1820 of *Lecture Notes in Computer Science*, pages 167–182. Springer Berlin / Heidelberg, 2000.

[25] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley-Interscience, 1973.

[26] J. Engblom. Analysis of the execution time unpredictability caused by dynamic branch prediction. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 152 – 159, may 2003.

[27] Junfeng Fan, Xu Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-art of secure ecc implementations: a survey on known side-channel attacks and countermeasures. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 76 –87, june 2010.

[28] Federal Communications Commission. Authorization and use of software defined radios. ET Docket No. 00-47, september 2001.

[29] B. Friedlander. System identification techniques for adaptive noise cancelling. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 30(5):699 – 709, oct 1982.

[30] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer Berlin / Heidelberg, 2001. 10.1007/3-540-44709-1_21.

[31] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin. Compatibility is not transparency: Vmm detection myths and realities. In *proceedings of Workshop on Hot Topics in Operating Systems*, 2007.

[32] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of Network & Distributed System Security Symposium*, 2003.

[33] Catherine Gebotys and Robert Gebotys. Secure elliptic curve implementations: An analysis of resistance to power-attacks in a dsp processor. In Burton Kaliski, etin Ko, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 235–250. Springer Berlin / Heidelberg, 2003.

[34] Christopher Hallinan. *Embedded Linux Primer*. Prentice Hall, 2007.

[35] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[36] IEEE Computer Society. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), 2003.

[37] Texas Instruments. Omap35x applications processor. Technical Reference Manual (SPRUF98M), 2010.

[38] Texas Instruments. Tps65950 integrated power management/audio codec. Data Manual (SWCS032E), 2011.

[39] Xuxian Jiang. Research webpage. Available at: *http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu3/*.

[40] Paul Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin / Heidelberg, 1996.

[41] P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proc. 19th Ann. Int'l Cryptology Conf. Advances in Cryptology: (CRYPTO 99)*, pages 388–397, 1999.

[42] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *ecurity & Privacy, IEEE*, 9(3):49 – 51, 2011.

[43] T. Laopoulos, P. Neofotistos, C. A. Kosmatopoulos, and S. Nikolaidis. Measurement of current variations for the estimation of software-related power consumption. *IEEE Transactions on Instrumentation and Measurement*, 52(4), August 2003.

[44] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[45] H. Mehta, R.M. Owens, and M.J. Irwin. Instruction level power profiling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 6, pages 3326 – 3329, May 1996.

[46] Microchip. Microchip Website. *http://www.microchip.com/wireless*.

[47] Microchip. Pic18f2525/2620/4525/4620 data sheet. DS39626E, 2008.

[48] Toshio Miyachi, Hiroki Narita, Hidekazu Yamada, and Hirohisa Furuta. Myth and reality on control system security revealed by stuxnet. In *SICE Annual Conference (SICE), 2011 Proceedings of*, 2011.

[49] A. K. Mok and L. Guangtian. Efficient run-time monitoring of timing constraints. In *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium*, June 1997.

[50] J. Newsom and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of IEEE Symposium on Security and Privacy*, 2005.

[51] S. Nikolaidis, N. Kavvadias, P. Neofotistos, K. Kosmatopoulos, T. Laopoulos, and L. Bisdounis. Instrumentation setup for instruction level power modeling. Technical report, Springer-Verlag, 2002.

[52] Vojin G. Oklobdzija. *Digital Design and Fabrication*. Boca Raton: CRC Press, 2008.

[53] Alan Oppenheim and Ronald Schafer. *Discrete-time Signal Processing*. Prentice Hall, 1989.

[54] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel. Power-analysis attack on an asic aes implementation. In *Information Technology: Coding and Computing, 2004. Proceedings International Conference on*, pages 546 – 552, 2004.

[55] Eric Peeters, Franois-Xavier Standaert, and Jean-Jacques Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration, the VLSI Journal*, 40(1):52 – 60, 2007. Embedded Cryptographic Hardware.

[56] Sarah Perez. Over 50 droiddream malware apps removed from android market. *The New York Times*, March 2nd, 2011.

[57] T. Popp, E. Oswald, and S. Mangard. Power analysis attacks and countermeasures. *Design & Test of Computers, IEEE*, 24:535 – 543, November 2007.

[58] T. Popp, E. Oswald, and S. Mangard. Power analysis attacks and countermeasures. *Design & Test of Computers, IEEE*, 24:535 – 543, November 2007.

[59] R.M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic. Power supply signal calibration techniques for improving detection resolution to hardware trojans. In *IEEE/ACM International Conference on Computer-Aided Design*, 2008.

[60] RowBoat. Android for Texas Instrument Devices. Available at: *http://code.google.com/p/rowboat/*.

[61] J.C. Ryoo, D.G. Han, S.K. Kim, and S. Lee. Performance enhancement of differential power analysis attacks with signal companding methods. *IEEE Signal Processing Letters*, 15:625 – 628, 2008.

[62] S. E. Chodrow and F. Jahanian and M. Donner. Run-time monitoring of real-time systems. In *Proceedings of the Twelfth Real-Time Systems Symposium*, Dec 1991.

[63] R.S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40 –48, sep 1994.

[64] Fresscale Semiconductors. Freescale Website. *http://www.freescale.com*.

[65] M. Sharif, W. Lee, W. Chui, and A. Lanzi. Secure in-vm monitoring using hardware virtualization. In *Proceedings of ACM conference on Computer and Communication Security*, 2009.

[66] Franois-Xavier Standaert, Loc van Oldeneel tot Oldenzeel, David Samyde, and Jean-Jacques Quisquater. Power analysis of fpgas: How practical is the attack? In Peter Y. K. Cheung and George Constantinides, editors, *Field Programmable Logic and Application*, volume 2778 of *Lecture Notes in Computer Science*, pages 701–710. Springer Berlin / Heidelberg, 2003.

[67] G.E. Suh, J.W. Lee, D. Zang, and S. Devadas. Secure program execution via dynamic information flow tracking. In *Proceedings of International Conference on Architectural Support for Programming languages and Operating Systems*, 2004.

[68] Song Sun, Zijun Yan, and J. Zambreno. Experiments in attacking fpga-based embedded systems using differential power analysis. In *IEEE International Conference on Electro/Information Technology*, pages 7 – 12, May 2008.

[69] Texas Instruments, Inc. TI Website. *http://www.ti.com*.

[70] J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley Publishing Company, 1974.

[71] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of des implemented on computers with cache. In Colin Walter, etin Ko, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 62–76. Springer Berlin / Heidelberg, 2003.

[72] X. Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic. Hardware trojan detection and isolation using current integration and localized current analysis. In *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, 2008.

[73] X. Wang, Y. Yin, and H. Yu. Finding collisions in the full sha-1. In *Proceedings of Crypto '05*, August 2005.

[74] N.H.E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley, 2nd edition, 1993.

[75] Anthony D. Whalen. *Detection of signals in noise*. Academic Press, 1971.

[76] J.M. Wing. A symbiotic relationship between formal methods and security. In *Computer Security, Dependability and Assurance: From Needs to Solutions, 1998. Proceedings*, pages 26 –38, 1998.

[77] Wytech. Wytech Website. Available at: *http://www.evbplus.com/index.html*.

[78] Yifeng Yang. Application note: An1204 microchip miwi p2p wireless protocol, 2008.